NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DEFENSE OF NAVAL TASK FORCES FROM ANTI-SHIP MISSILE ATTACK

by
James R. Townsend

March, 1999

Thesis Advisor: James G. Taylor
Second Reader: Arnold A. Buss

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 1999	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE DEFENSE OF NAVAL TASK FORCES FROM ANTI-SHIP MISSILE ATTACK	**5. FUNDING NUMBERS**
6. AUTHOR(S) Townsend, James R.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

The quantity, capability, and availability of Anti-Ship Missiles (ASMs) pose a significant threat to the safe operation of United States Naval Forces in the waters off of potentially hostile shores. Potential adversaries continue to improve their ability to attack our ships, requiring that we constantly analyze our defenses against such attacks. Existing computer models and simulations, do not provide force commanders or naval analysts with an adequate tool to properly evaluate the threat and the best ways to minimize it. This thesis has developed such an analysis tool, called the Anti-Ship Missile Defense (ASMD) model. It allows for analysis to be performed from an entire task force perspective, modeling the entire process by which ASMs select their targets and the methods by which the defending escorts assign defensive fire. Effective Screen Design and Defensive Firing Policy is a large and complex problem, but exploratory analysis using ASMD has yielded useful insights. In ASMD, moving objects are more fully rendered, featuring smooth acceleration, turning and altitude change features. In support of these complicated moving entities, a highly capable mathematical library was created to solve the resulting equations of motion. The software components and architecture developed for ASMD provide significant flexibility and reuse potential for future analysts.

14. SUBJECT TERMS Aegis Defense System, Standard Missile, Cruise Missile, Java Simulation System	**15. NUMBER OF PAGES**
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

DEFENSE OF NAVAL TASK FORCES FROM ANTI-SHIP MISSILE ATTACK

James R. Townsend
Lieutenant Commander, United States Navy
B.S., Marquette University, 1984

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
March 1999

Author: ______________________________
James R. Townsend

Approved by: ______________________________
James G. Taylor, Thesis Advisor

____
Arnold A. Buss, Second Reader

____
Richard E. Rosenthal, Chairman
Department of Operations Research

ABSTRACT

The quantity, capability, and availability of Anti-Ship Missiles (ASMs) pose a significant threat to the safe operation of United States Naval Forces in the waters off of potentially hostile shores. Potential adversaries continue to improve their ability to attack our ships, requiring that we constantly analyze our defenses against such attacks. Existing computer models and simulations, do not provide force commanders or naval analysts with an adequate tool to properly evaluate the threat and the best ways to minimize it. This thesis has developed such an analysis tool, called the Anti-Ship Missile Defense (ASMD) model. It allows for analysis to be performed from an entire task force perspective, modeling the entire process by which ASMs select their targets and the methods by which the defending escorts assign defensive fire. Effective Screen Design and Defensive Firing Policy is a large and complex problem, but exploratory analysis using ASMD has yielded useful insights. In ASMD, moving objects are more fully rendered, featuring smooth acceleration, turning and altitude change features. In support of these complicated moving entities, a highly capable mathematical library was created to solve the resulting equations of motion. The software components and architecture developed for ASMD provide significant flexibility and reuse potential for future analysts.

THESIS DISCLAIMER

The reader is cautioned that the computer programs developed in this research may not have been exercised for all cases of interest. While every effort as been made, ithin the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

List of figures

EXECUTIVE SUMMARY

The quantity, availability, and capability of Anti-Ship Missiles pose an ever increasing threat to the safety of United States Navy forces. Current Anti-Ship Missile Defense systems are deemed adequate, but the future is more uncertain. Further analysis of Missile Defense formations and tactics is clearly necessary in order to see the Navy safely into the twenty first century.

Current naval combat models do not provide sufficient analysis capacity to evaluate competing tactical alternatives. High-resolution models focus solely on defense of a single ship and cannot be realistically extended to analyze the problem of screen defense as a whole. Aggregated campaign models do not attempt to model the process by which Anti-Ship Missiles detect and attack ships. A new Combat Model is clearly necessary to conduct meaningful analysis of Screen Defense against Anti-Ship Missiles.

The Anti-Ship Missile Defense (ASMD) simulation was created to correct these existing model shortcomings. It allows for more realistic object movement simulation, owing to a substantial supporting mathematical package. This enhanced realism enables for more accurate simulation of the missile attack. Each incoming missile ’sees’ the screen of ships from its own unique perspective, owing to the geometry of the screen design, the size of the ships, the missile’s altitude, and direction of motion.

Exploratory analysis utilizing the ASMD model has revealed considerable insights into screen design and defensive firing policy. It is possible to prevent enemy missiles from detecting and homing on the High Value Unit (HVU) by the careful choice of escort

ship locations. Analysis has also demonstrated the superiority of a Shoot-Shoot-Look firing policy, over Shoot-Look-Shoot, when the formation is attacked by moderate sized enemy missile attacks (10 - 50 missiles).

The ASMD model is a powerful tool, and its uses extend beyond the analysis of screen defense problems examined so far. The model can evaluate the threat posed by multi-axis missile attacks, the impact of decoy, and other tactics. It can be used, to plan missile attacks against enemy ship formations.

This study must be regarded as a first step toward enhancing the safety of our ships at sea. Additional analysis will be necessary in order to achieve the goal of minimizing the threat of Anti-Ship Missiles to our naval forces.

ACKNOWLEDGEMENTS

The author would like to express his thanks to Dr. James Taylor and Dr. Arnold Buss. Their unflagging support, technical advice, and encouragement to think ’outside the box’ were instrumental in the completion of this work. Dr. Taylor posed an interesting question regarding the suitability of a hierarchy of models approach to naval warfare. He allowed the focus of the thesis to evolve into the development of a new combat model, when it was found that existing simulations were inadequate to the task at hand. Dr. Buss provided considerable technical advice into the development of a complex and powerful analysis tool. His timely suggestions were deeply appreciated.

Finally, the author would like to express gratitude for the support and patience of his wife, Joyce. Her support and forbearance exhibited over nearly 1500 hours of simulation development were vital to the success of this project.

I. INTRODUCTION

Ships of the United States Navy routinely operate in hostile waters throughout the world. They do this in furtherance of the National policy goals of the United States Government, working in close proximity to potential adversaries that may possess weapon systems capable of attacking our forces at sea. The current pattern of operations indicates that this trend will not change in the foreseeable future.

The hazard presented by Anti-Ship Missiles (ASM) to our Naval Forces is on the increase. Currently, 13 nations (not including the United States and its NATO allies) possess an organic ASM production capacity. A further 15 nations are developing this capability. The dissolution of the Soviet Union has resulted in an increase in the export of that nations weapon technology and hardware. Other major arms supplying nations, especially France and China, are making substantial improvements to the capabilities of their missile systems. While the threat of ASM attack against the United States Navy may not be overwhelming at the moment, the increase in availability and capability of these weapons necessitates our constant analysis and development of defensive systems and tactics so that we can operate with the impunity that we currently enjoy.

A. MOTIVATION

To ward off the ASM threat, our Navy has employed a multi-pronged defensive policy:

1. Avoidance of known threats.

We try to avoid entering within range of known threat weapon systems.

2. Minimize information provided to an adversary.

We minimize the likelihood of an adversary gaining accurate targeting information against our ships by the maintenance of Carrier Air Patrols (CAP), that serve to keep adversarial aircraft beyond arm’s reach. We maintain a vigilant watch over the surface picture identifying all ships in the region. We closely monitor the undersea threat, by aggressive Anti-Submarine Warfare (ASW) tactics. All of these minimize the risk of the adversary gaining enough accurate targeting data to mount a serious threat against us.

3. Layered defensive systems.

These defenses consist of search sensors that can detect incoming aircraft and missiles at long range, Surface to Air Missile (SAM) systems that can track and engage the airborne adversary, Electronic Warfare systems that can jam or confuse incoming missiles, and lastly, high speed gun systems that may shoot down incoming weapons at very close range.

These tactics may not prove to be sufficient in the future. Our forward operations will make it more likely for the enemy to locate our ships. Shallow waters may defeat our effective ASW efforts. Mobile ashore sensors and weapon systems may escape our surveillance and pose a real and substantial threat to our forces. Multi-axis missile attacks may saturate the defensive systems and render the ships susceptible to damage.

Anti-Ship Missiles (ASMs) may pose the single greatest threat to the safe operation of our ships in forward areas. The numbers, sophistication, and availability of

ASMs serve to ensure that this substantial threat will never diminish. The United States Navy will need to continuously analyze the threat of ASMs and the proper defensive measures to counter it.

In light of current operations in the Persian Gulf, the following questions require immediate analysis:

1) What escort ship spacing and orientation patterns are effective at minimizing the threat posed by mobile Iraqi Silkworm missile sites?

2) Is there a benefit to conducting a Shoot-Look-Shoot (SLS) defensive firing policy, opposed to Shoot-Shoot-Look (SSL), to counter an incoming missile attack?

It is beyond the scope (and classification) of this thesis to be able to definitively answer these questions. Missile and Radar performance data is classified, and enemy planning for missile employment is an unknown. Nevertheless, there is considerable insight that can be provided by the Anti-Ship Missile Defense (ASMD) model based solely on unclassified data, operating experience, and reasonable assumptions. Such an analysis will be conducted in Chapter IV of this report.

B. BACKGROUND.

Existing analysis methods consist of a handful of computer models that simulate the ASM battle. These models are inadequate, however, in providing meaningful analysis methods for the purpose of examining screen defense. Until quite recently, the limitations of computer speed and memory largely restricted analysis to the defense of a single ship against incoming missiles. The results from single ship defense could be extrapolated to each ship in the formation, because it was assumed (or insufficiently modeled) that the incoming missile raid would be dispersed uniformly among the ships present (e.g. a Salvo of 25 missiles against a formation of 5 ships would result in 5 missiles per ship. If each ship could successfully defend against 5 missiles, no damage would occur, etc…).

The state of the art in ASMD modeling consists of only a few models. These range from the Single Ship Air Defense Model (SSADM), which simulates the defensive firing capabilities of a solitary ship that is exposed to enemy missile fire, to several highly aggregated campaign simulations (such as the Joint Theater Level Simulation [JTLS] and the Integrated Theater Engagement Model [ITEM]).

SSADM was developed to emphasize the defensive power of a single warship. SSADM does a credible job of analyzing the process by which an Aegis Cruiser conducts self-defense. It also models defensive shots against missiles not targeted at the cruiser. It does not, however, conduct run-time analysis of the flight and homing patterns of the incoming missiles. The dispersion of enemy fire is set at run-time by the analyst and is not a function of the actual geometry of the defensive screen. SSADM cannot be extended, however, to fully examine situations in which more than one ship is present and may be targeted.

Using these limited capability High-resolution models as their inputs, Joint Campaign Models, predictably, tend to poorly emulate the impact of ASM weapons against naval forces. For example, the Integrated Theater Engagement Model (ITEM) treats missile attacks against ships as a purely Monte-Carlo evolution. The incoming raid is divided evenly over the defending ships, and each ship is adjudged to receive a number of hits in direct proportion to the number of missiles assigned to it. This is essentially an extension of the SSADM methodology. The Joint Theater Level Simulation (JTLS) uses this same logic to adjudicate naval combat. These aggregated models cannot be used to analyze the impact of screen design and defensive firing posture because they treat all screens and firing policies as identical items. These models do not consider the geometrical differences and effect of firing policy.

As discussed above, the process by which an ASM detects and attacks a target selection has not been effectively modeled. In addition to the simulations discussed above, most ongoing analysis at the Naval Postgraduate School (NPS) starts the ship defense problem at a point after the incoming missiles have selected their targets. As a result, these models also focus on individual ship defense, as opposed to defense of the entire screen against an entire missile attack.

In light of these limitations, it becomes evident that existing models and simulations are insufficient in providing a method of analysis. The development of a new and effective analysis became a necessary precursor to analyzing the problems at hand.

In this thesis, we attempt to redress these simulation limitations by developing a model that more effectively emulates the problems of ASM Defense. The model employs a more rational missile distribution pattern that is based on the actual geometry presented to the incoming missiles. Additionally, this new model will feature data generation capabilities that will allow for ease of incorporation into higher level models (using a hierarchy of models approach).

It was decided to utilize the Java programming language in the creation of the new model. Java was a logical choice, because the author could not foresee the precise computer/operating system arrangement that would be used to conduct additional analysis, it was desirable to use a platform-independent programming language. Additionally, there exists a tremendous body of existing Java simulation components developed at NPS. Former students (LT Kirk Stork, USN and MAJ Arent Arntzen, RNAF) have developed these components in conjunction with Visiting Professor Arnold A. Buss. The libraries that were developed, Simkit and Modkit, respectively, have served as the backbone for the development of the ASMD model.

In Chapter II, we will outline the requirements for this new analysis tool and sketch out its desired functions. In Chapter III, we will discuss the functions and combat resolution logic of the ASMD model. In Chapter IV, we will conduct an analysis based on several simulation runs, and examine the types of results that can be made available to the analyst. In Chapter V, we will outline other potential uses for the ASMD model, as well as summarizing the results of this thesis.

II. DEVELOPMENT OF A NEW ANALYSIS TOOL, THE ASMD MODEL

When contemplating the creation of a new combat model, it is important to identify the purposes of that model. These purposes may be manifold, and can include; (1) Hardware Acquisition, in which the new system (or additional purchases) are evaluated for their comparative worth. (2) Force Structuring, in which the force is shaped to incorporate the correct ratio of weapon systems of the right types. (3) Tactical Development, in which non-lethal simulation can identify potential strengths and weaknesses of certain tactics. (4) Capability of Forces, where the ability of the force to accomplish missions in theater is evaluated.

The ASMD model developed for this thesis can directly, or in conjunction with other tools, be used for each of these purposes.

Now that we have stated the purpose of the new model, and have identified weaknesses or gaps in existing models, we will define the battle space that affect the outcome of combat within that realm.

A. NAVAL TASK FORCE.

The first, and most pertinent, entity to define is the naval task force (TF). Simply stated, a TF is a collection of naval combatants and auxiliaries that are grouped together for the accomplishment of one or more missions. The individual ships function together as a team to provide mutual support and defense against opposition to assigned missions. These ships are typically arrayed into a formation, called a screen, in which the most valuable and important units (termed high value unit, or HVU) are surrounded and protected by numerous escorting vessels. Within the screen, the escort ships are stationed in sectors away from the HVU, as shown in Figure 1.

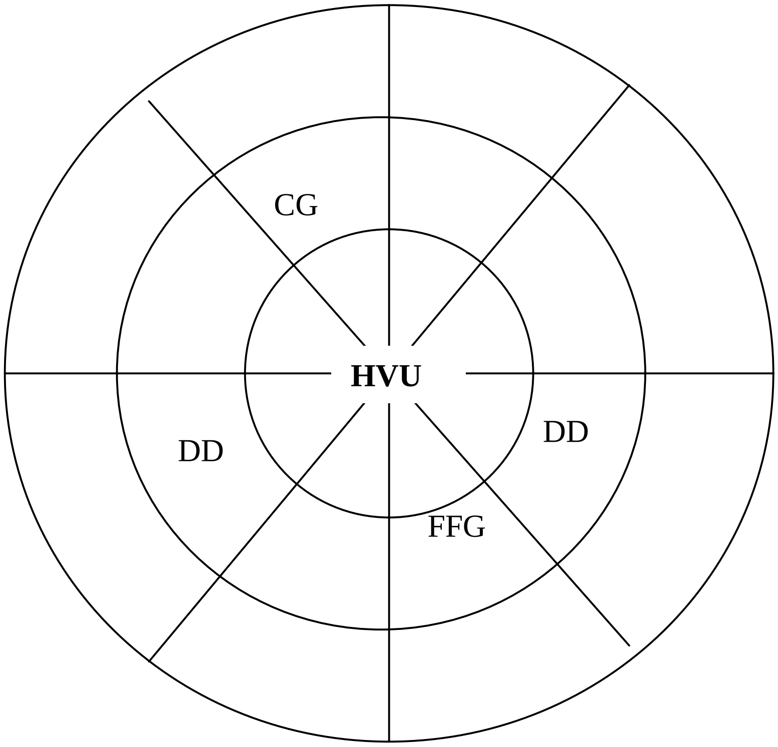


Figure 1 A Typical Screen

B. DETECTION OF THE THREAT.

The ships of the TF possess numerous sensors that range from passive (listen-only) Electronic Warfare and Sonar systems, to active (emitter) Sonar and Radar systems. For the purposes of this thesis, we will confine our interest to Radar sensors only. Radar systems feature a maximum theoretical range, which is a function of their power output, Pulse Repetition Frequency, and assumed target radar cross-section. Radar systems are mounted high above the waterline so as to maximize the distance to the horizon.

C. DEFENSIVE WEAPONRY.

Most ships also host defensive missile and gun systems that can be used to engage enemy aircraft and missiles. Since we are primarily concerned with missile defense by missiles in this thesis, we will treat defensive gunfire and electronic warfare systems in an aggregate fashion.

D. ASMD SYSTEM.

The ASMD system consists of the launching platform (which may be an aircraft, ship, or shore site), and the individual missiles themselves. These ASM missiles are launched in the general direction of the TF that has been targeted.

E. WHAT’S AHEAD.

In this section we have discussed the inadequacy of current analysis tools as they apply to the enhancement of screen defenses. We have briefly defined the ASMD battlefield and the basic limitations of the new model. In the next chapter, we shall discuss the operation of the ASMD model, focussing on its more salient features.

III. LOGIC OF THE ASMD MODEL

In this section we will examine the operation of the ASMD model. We will focus only on the most salient points concerning the model functions. A complete User's Guide, and other more descriptive work is planned for the future, but is beyond the scope of this thesis.

Combat in the ASMD model is conducted between entities at the Composite Unit level. A Composite Unit, for the purposes of this thesis, is a group of special purpose functional components that operate together. The meaning of this term will become clear in the example below. Using the premise of small reusable object programming, these composite units are created from several smaller components that seek to model the precise behaviors of the composite unit. In the discussion that follows, we will break apart this Composite Unit into its individual components, and briefly explain the functioning of each.

It may be useful to have in mind a specific type of Composite Unit with which almost everyone will be familiar, the automobile. With this image in mind, lets look at how the functions of this Composite Unit could be divided into logical component groups.

A. AUTOMOBILE COMPONENT FUNCTIONS

A human driver controls an automobile. The driver manipulates the movement controls of the automobile, such as the steering wheel, accelerator and brake pedals, to cause the automobile to proceed to destinations that the driver would like to visit. The automobile has a windshield and mirrors that serve to allow the driver to see the road surface, information signs, other automobiles, and hazards that must be sensed in order to proceed from one place to the next. The automobile has turn signals, headlights, and a horn that allow its driver to signal, and thus interact, with other automobiles on the road.

Within this small example, let’s now try to compartmentalize the functions that we have identified. There appear to be four primary divisions of functionality.

There is a controller, namely the driver, that directs the operation of all movement controls for the automobile.

There is a movement element, consisting of the engine controls, engine and drivetrain, steering wheel, and tires that cause the automobile to travel from place to place in response to the driver’s direction.

There are sensing elements, consisting of the windshield, windows, and mirrors, in addition to the drivers eyes and ears, that allow the driver to evaluate the current environment and make modifications to the operation of the movement element.

Finally, there are interaction elements, consisting of the horn, turn signals, and headlights, that can inform other automobiles (and their drivers) about the actions and intentions of this automobile. Figure 2 illustrates the arrangement of these groups of functionality (or components).

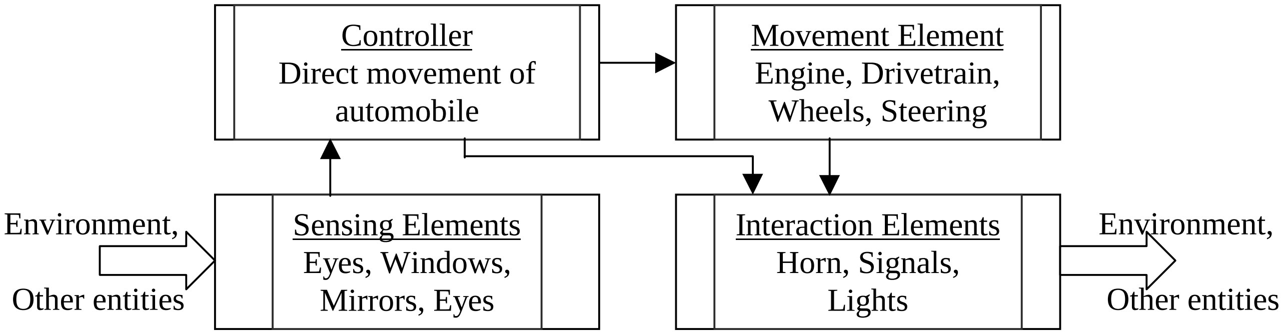


Figure 2 Automobile Functional Components

Most entities on a battlefield, such as ships, missiles, or tanks, can be seen to contain most, if not all, of these types of functionality. They each have a controller (that may be resident in each unit, or may be lumped together to control the functionality of many units), a mover of some sort, sensors, and interactors (such as guns) This obvious compartmentalization scheme has been exploited by others (such as Lt. Stork and Maj. Arntzen) and was seized upon in the development of the ASMD model, as well.

Now that we've identified the functional components that are contained within a composite unit, we will briefly discuss the characteristics of the specific components developed for use in the ASMD model.

1. Controller

a) MoverBrain

There are many aspects of naval entity operations that can be seen to be controlled. Of primary concern are the control of movement, and the management of defensive systems.

An object called the MoverBrain accomplishes control of movement. There are two versions of this object, the first is designed for use in controlling objects that are restricted to movement along the surface of the earth (ocean). These are termed 2–Dimensional (or 2D movers), and consequently, this controller is called the MoverBrain2D. Aircraft, submarines, and Missiles travel in three dimensions (3D movers), and are controlled by MoverBrain3D objects.

The MoverBrain is a relatively simple object. It stores the list of destinations and times that the composite unit will traverse, and tells the Movement Element exactly how to proceed from one point to the next. Upon start of movement (or when an intermediate destination is reached or new orders are received), the MoverBrain examines the movement capabilities of the Movement Element, and plans the best series of maneuver operations necessary to cause the Composite Unit to arrive at the next location at the correct time. The MoverBrain sends a message to the Movement Element telling it exactly what turn rate to assume, (and for how long) so that the Composite Unit will travel in the correct direction. The MoverBrain directs the Movement Element at what rate to change speed, and for what duration, so that the correct and necessary speed will be achieved. The MoverBrain3D directs the 3D Mover Element at what rate and for how long, to change altitude. Figure 3 summarizes these MoverBrain functions.

MoverBrain

Respond to sensors and Task force Controller

Calculate Necessary Maneuver Parameters

Turn Rate	Acceleration Rate	Climb Rate
Turn Duration	Acceleration Duration	Climb Duration

Direct: Movement Element to execute necessary maneuvers to accomplish movement orders.

Figure 3 Mover Brain Functions

b) FireDistributor (cursory introduction)

An object called the FireDistributor manages the execution of defensive fire against incoming missiles. There is one FireDistributor per side in the simulation, and the functionality of this object will be discussed in greater detail, later.

2. Movement Element

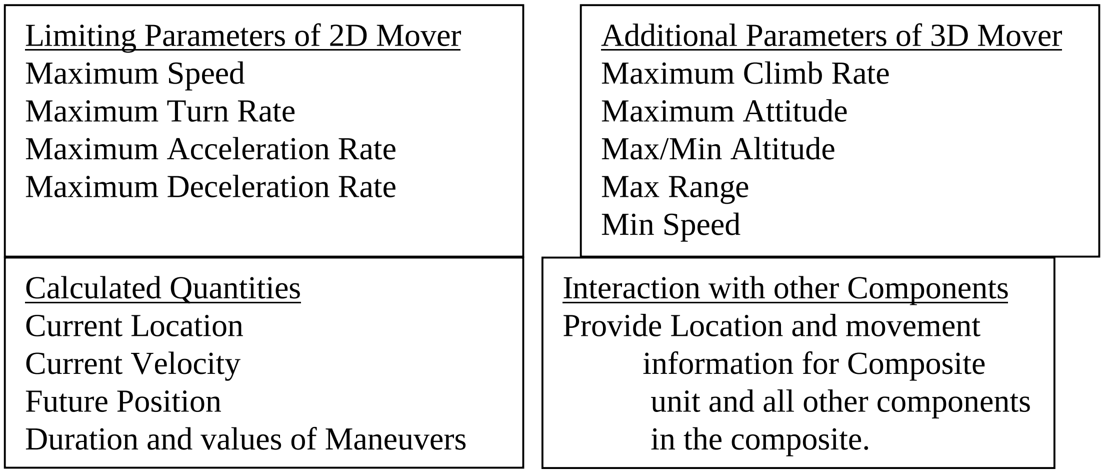
a) FullMover

An object called the FullMover manages movement of the Composite Unit. Depending on the type of movement desired for the Composite Unit (2D or 3D), there are FullMover2D and FullMover3D to accomplish it.

The FullMover is a more complicated object than the MoverBrain. The FullMover responds to the movement orders of the MoverBrain but also must provide instantaneous reports and updates to the precise location of the Composite Unit within the space of the battle field. The FullMover is the storehouse of all movement limits (maximum and minimum speed, altitude, turn rate, climb/dive rate, acceleration and deceleration rates, et. al.) The FullMover must inform other components, using the Referee, which will be discussed later, of changes to the direction, speed, or altitude of the Composite Unit. The FullMover also contains 'forecasting' logic that allows it to tell other components where it will be at any time in the future (based on its current movement parameters). The reasons for this prediction capability will be made clear later, in our discussion of Detection. Figure 4 summarizes the parameters and functions of the FullMover Component.

3. Sensing Elements

Sensing capability for the Composite Unit is provided by any number of Sensor Objects. In the current version of the ASMD model, there are only active Radar systems employed, although plans exist to extend the model to incorporate passive and multi-modal sensors (such as Electronic Surveillance Measures [ESM], and Sonar systems).



Limiting Parameters of 2D Mover
Maximum Speed
Maximum Turn Rate
Maximum Acceleration Rate
Maximum Deceleration Rate

Additional Parameters of 3D Mover
Maximum Climb Rate
Maximum Attitude
Max/Min Altitude
Max Range
Min Speed

Calculated Quantities
Current Location
Current Velocity
Future Position
Duration and values of Maneuvers

Interaction with other Components
Provide Location and movement information for Composite unit and all other components in the composite.

Figure 4 FullMover Functionality

Each Sensor contains parameters that limit the maximum range at which targets can be detected, the type of target it can detect (2D or 3D), the rate at which detections can occur, and the maximum number of targets it is capable of simultaneously tracking. The Sensor broadcasts messages to the referee components whenever it detects or loses a target, and when it changes from being ‘on’ to ‘off’ or vice-versa. Again, there may be multiple Sensors on each Composite Unit. Figure 5 summarizes these functions.

Limiting Parameters of Sensor
Maximum Range
Maximum Detection Rate
Maximum Number Targets

Interaction with other Components
Provide Sensor Status and list of current targets.

Figure 5 Sensor Functionality

4. Interaction Components

Interaction capability for the Composite Unit is provided by any number of Launcher Objects. In the current version of the ASMD model, there are only Missile systems actually utilized. Plans exist to incorporate Guns, Chaff, Torpedoes, etc… into future versions. Each Launcher possesses properties that control what the maximum launch rate is, and quantity and types of missiles that can be launched. The launcher responds to orders from the FireDistributor (which tells the Launcher exactly how many and what type of missiles to shoot, and at which target(s)). At launch time, the Launcher broadcasts the launch to the referee components so that they are aware of the existence of a new Composite Unit on the battlefield. Figure 6 summarizes the launcher capabilities.

Limiting Parameters of Launcher	Interaction with other Components
Maximum Fire Rate	Respond to Launch Orders.
Maximum Guidance Capacity	Create new missile entities.

Figure 6 Launcher Functionality

5. Assembling the Composite Unit

The Composite Unit is constructed by an object called the TacticalUnit. This object collects the identity of each low-level component that is added to the Composite Unit, and stores this identity for easy reference. The TacticalUnit object also connects each of the components to each other so that they may function as one big entity, despite being totally separate in their functionality. (e.g. this means that other components can ask a specific radar on the Composite Unit for the radar's current location. The radar will have this information automatically forwarded by the Composite Unit's FullMover. Orders to the Composite Unit to move to a new location will be forwarded automatically to the MoverBrain inside of the Composite Unit). For convenience, there are numerous specific instances of particular Composite Units existing already within the ASMD project. These include entities such as specific ships, missiles, and shore-based launchers. Nothing precludes other users from creating their own Composite Units as required.

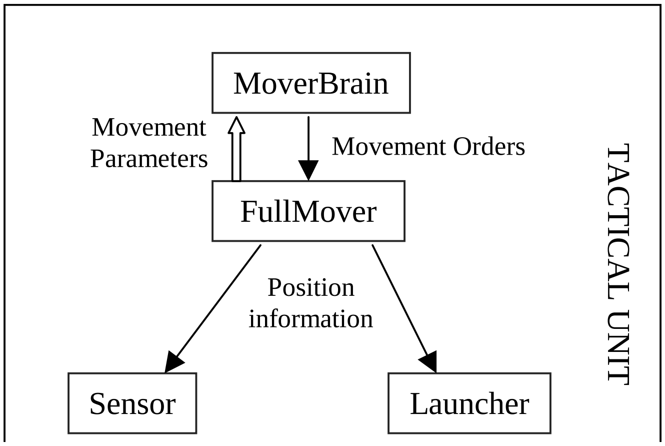
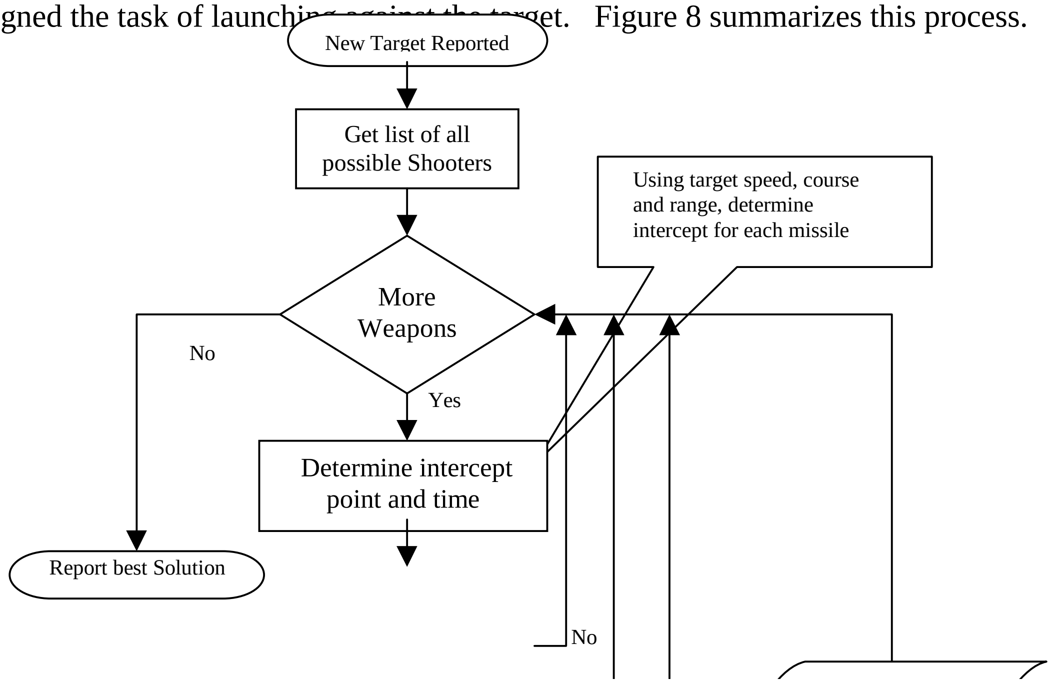


Figure 7 Tactical Unit Internal Connections

6. FireDistributor (detailed description)

We will now discuss the FireDistributor in greater detail. As has been mentioned before, there is one FireDistributor per side (although future versions of ASMD will feature one FireDistributor per screen since there may be more than one screen deployed), and this is the object that performs all of the decision logic necessary to allocate defensive fire against enemy forces. Whenever a sensor on its side reports that it has detected a new target, the identity of the target is passed to the FireDistributor. The FireDistributor searches for this target on its master list of targets for the force. If this is a new target, the FireDistributor will allocate fire against the target. In a nutshell, this is accomplished by canvassing each of the capable weapon systems within the force and determining which system can intercept the target first. The system that could conceivably hit the target first is assigned the task of launching against the target. Figure 8 summarizes this process.


New Target Reported
Get list of all possible Shooters
Using target speed, course and range, determine intercept for each missile
More Weapons
No
Yes
Determine intercept point and time
Report best Solution
No

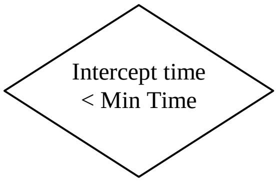


Figure 8 Defensive Fire Logic

B. REFEREE COMPONENTS

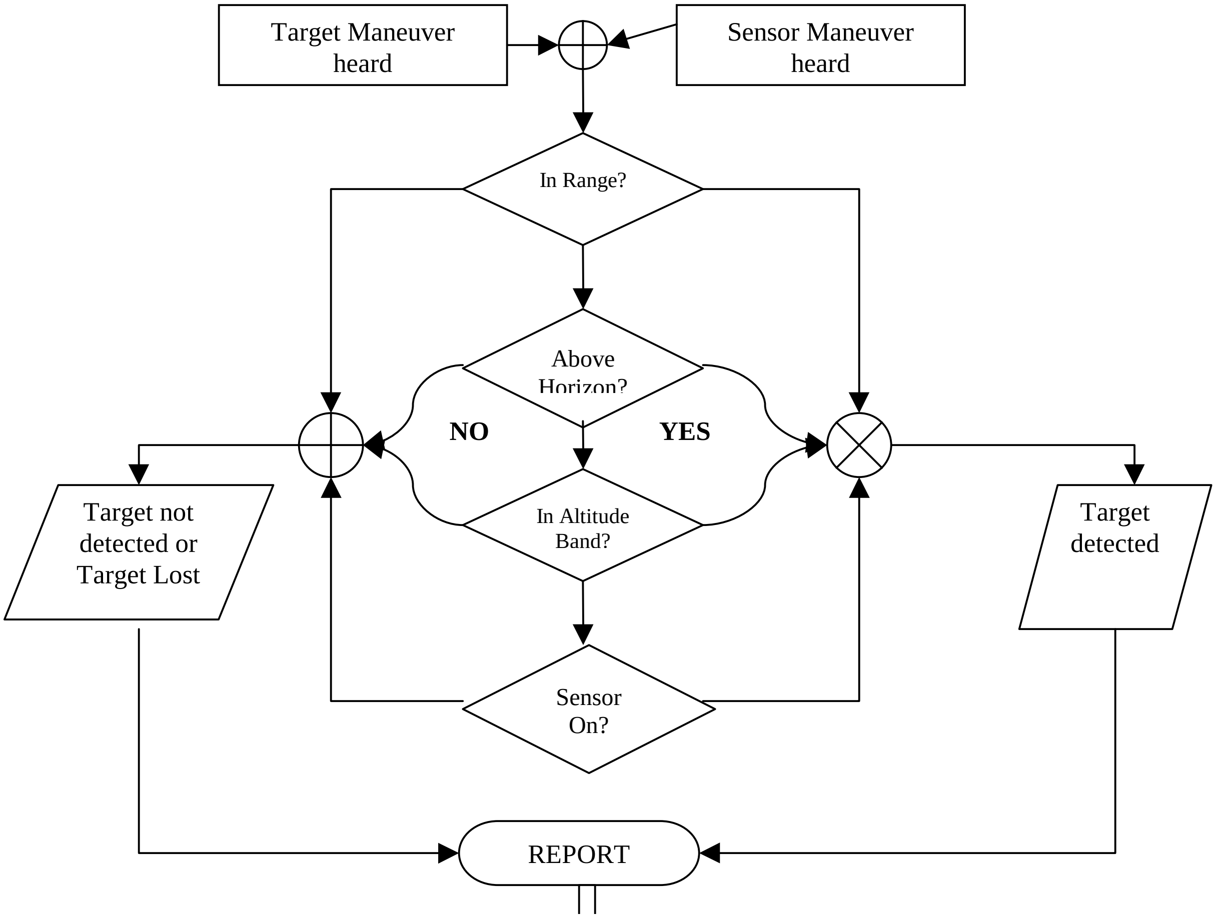
We have made repeated references to Referee Components. It is now time to discuss these important items. There are three separate components that perform necessary Referee tasks: the Register, the Mover Sensor Mediator, and the Missile Target Mediator.

1. Register.

The Register is simply master list of all of the Composite Units on each side. The Register does not adjudicate any interactions of its own accord, but rather creates and destroys instance mediators that accomplish the adjudication tasks. The Register reacts to the addition of a new Composite Unit (such as the launching of a missile) by first looking at the new unit. If it is a missile, and has a target already assigned (such as a SAM that is being guided against an ASM), it will create a Mediator between the missile and its target. If the new unit does not have a target, the Register will create a Mediator between every sensor on the Composite Unit and each opposing force Composite Unit that could be detected by that sensor (provided that the sensor is 'on'). Once all of the new Unit sensors are connected, the Register creates a mediator between all opposing force sensors that are 'on' (and could detect the new Unit), and the new unit.

2. Mover Sensor Mediator.

As the name implies, this is a component that resolves issues between a single moving target and a single detection capable sensor. This mediator does not belong to any side, but is an unallied component. In this capacity, it can serve as an honest broker in examining the true battle picture. It can then evaluate if and when the mediated target becomes detected or undetected by the mediated sensor. Figure 9 summarizes this process. The Mover Sensor Mediator accesses the mathematical features provided by several 'helper' objects. Each of these will be discussed later.



3. Missile Target Mediator.

This is a component that adjudicates the interaction between missiles that are homing against a target. The Mediator listens to the movement of the target, and directs the homing (or guided) missile to adjust its flight to intercept the target. The Mediator schedules the detonation of the missile when it is finally close enough to the target to hit it, and evaluates the outcome of this detonation.

a) Combat Results

There are three outcomes possible when a missile terminates near its target. The first is that the missile explodes too far away from the target to damage it (a miss). In this case, the missile is destroyed, but the target is unaffected. If the target is an ASM (or other enemy aircraft) the side that owned the missile hears the miss, and schedules another launch against the incoming enemy unit. A second outcome is that the missile may hit the target. In this case, the missile is destroyed and the target registers a hit against it. If the target is a missile or aircraft, it too is destroyed and removed from the battle. The last outcome is that the missile may be destroyed by defensive fire emanating from the target, or may be confused by Electronic Warfare (such as chaff or jamming). In this case, the missile is destroyed, but is treated as a miss in all other respects. This last result is only possible when the target is a warship. Each outcome is determined by Monte Carlo techniques, with a fixed probability of each result occurring.

C. Supporting Objects.

2. Event Step versus Time Step Methodology

The ASMD model uses Event-Step methodology. What this means is that calculations are performed, and situations evaluated only when conditions actually change. Checks are not made at fixed time intervals (as in Time-Step methods). The primary disadvantage of the latter (Time-Step), is that event sequencing may not be correct and the outcome of the battle may depend greatly on the size of the time step selected. The choice of Event Stepping avoids this disadvantage, but may sometimes result in a much larger calculation burden to ensure that events are properly scheduled.

In order to minimize the size of individual components, much of the mathematical functionality is removed to supporting objects. Each of these can be created and destroyed rapidly, thereby minimizing the amount of computer memory required for execution of the model.

The decision, made early on, to incorporate more fully rendered movers (featuring acceleration and smooth turning) led to problems that were not fully appreciated at the time that decision was made. The equations of motion for objects that can turn and accelerate become high order polynomials, the solution of which must be accomplished quickly during simulation runs. The solution of these polynomials (4th order and higher) necessitated the development of an extensive mathematical package. We will next discuss the development and functions of the AdvancedMath Package in detail in the following section.

The discussion that follows is not essential to understanding the analysis contained in Chapter IV, and the reader may safely skip the remainder of this chapter if he is not interested in the Math Package development. We'll start with a review of the mathematics package, and then explore how it is used by the ASMD model.

2. MATHEMATICS PACKAGE

It was necessary to examine the movement of objects in the defined battlefield, and explore the limitations of the existing Java.Math library. Simple linear movers, in which speed and course are constant, can readily report their movement quantities. If we view the world in two dimensions (X and Y), if an object starts at a point P_o, with coordinates (X_o, Y_o), the position of the object at any given time (t), can be easily calculated from its velocity (V_x, V_y) as Equation (1) demonstrates:

$$\mathbf{X(t) = X_o + V_x * t,} \qquad \mathbf{(1)}$$
$$\mathbf{Y(t) = Y_o + V_y * t}$$

To find the time that an object will arrive at a given location, given its starting point and velocity (and assuming that it is in fact proceeding at the correct speed and along the correct heading), all one need do is solve a series of linear equations,

Range = sqrt(dY^2 + dX^2)

Time = range/speed (where speed = sqrt($V_x{}^2$ + $V_y{}^2$))

This becomes significantly more challenging, however, when we examine the case where a mover changes course and speed smoothly (as opposed to instantly). If an object is accelerating (uniformly), its position is found by:

Let

accX represent the acceleration rate in the x direction
accY represent the acceleration rate in the y direction
velX represent the initial velocity in the x direction
velY represent the initial velocity in the y direction
xo represent the initial x position
yo represent the initial y position

$$x_t := xo + velX \cdot t + \frac{1}{2} \cdot accX \cdot t^2 \qquad (2)$$

$$y_t := yo + velY \cdot t + \frac{1}{2} \cdot accY \cdot t^2$$

If, instead, the object is turning,

Let

t represent time (and s be a dummy variable representing time), and
v represent the velocity
Co represent the starting course of the mover
rate represent the turn rate of the mover

The equations of motion for the mover become

$$x_t := xo + \int_0^t v \cdot \sin(Co + s \cdot rate)\, ds \qquad (3)$$

$$y_t := yo + \int_0^t v \cdot \cos(Co + s \cdot rate)\, ds$$

$$x_t := xo - \frac{\cos(Co + rate \cdot t)}{rate} \cdot v + \frac{\cos(Co)}{rate} \cdot v$$

$$y_t := yo + \frac{\sin(Co + rate \cdot t)}{rate} \cdot v - \frac{\sin(Co)}{rate} \cdot v$$

where rate is the rate of the course change and v_t is the speed at time t.

Taking this one step further, if the object is both accelerating and turning, simultaneously,

Let

t represent time
Vo represent initial velocity
Co represent starting course
yawRate represent the turn rate
accRate represent the acceleration rate

The equations of motion for the mover become

$$v_t := vo + \int_0^t \text{accRate}\, ds \qquad (4)$$

$$v_t := vo + \text{accRate}\cdot t$$

$$x_t := xo + \int_0^t (vo + \text{accRate}\cdot s)\cdot \sin(Co + s\cdot \text{rate})\, ds$$

$$x_t := \left[xo - \frac{(vo\cdot\cos(Co + \text{yawRate}\cdot t)\cdot\text{yawRate} + \text{accRate}\cdot\cos(Co + \text{yawRate}\cdot t)\cdot\text{yawRate}\cdot t - \text{accRate}\cdot\sin(Co + \text{yawRate}\cdot t))}{\text{yawRate}^2} \right] \cdots$$
$$+ \frac{vo\cdot\cos(Co)\cdot\text{yawRate} - \text{accRate}\cdot\sin(Co)}{\text{yawRate}^2}$$

$$y_t := yo + \int_0^t (vo + \text{accRate}\cdot s)\cdot\cos(Co + s\cdot\text{yawRate})\, ds$$

$$y_t := \left[yo + \frac{(vo\cdot\sin(Co + \text{yawRate}\cdot t)\cdot\text{yawRate} + \text{accRate}\cdot\sin(Co + \text{yawRate}\cdot t)\cdot\text{yawRate}\cdot t + \text{accRate}\cdot\cos(Co + \text{yawRate}\cdot t))}{\text{yawRate}^2} \right] \cdots$$
$$+ \frac{vo\cdot\sin(Co)\cdot\text{yawRate} + \text{accRate}\cdot\cos(Co)}{\text{yawRate}^2}$$

With this complicated movement scheme, trying to predict the specific maneuvers necessary (acceleration and turn rates and times) becomes significantly more challenging. Still more challenging, is the resolution of interactions between two movers that feature this type of movement. It became obvious that if we wanted to retain event-step methodology, we would need to be able to solve equations that were high order polynomials. These worked out to 4th order polynomials in the event of accelerating movers, only, and 16th order for turning movers (due to Power series expansion of the sine and cosine functions). In Java, there was no direct method (nor is there a convenient method in any other programming language) to solve for all of the roots that could be generated, including Complex roots. The AdvancedMath package seeks to solve these limitations.

a) ***Polynomial Equation***

The equation itself is simply an array of coefficients listed in descending power order. The Equation object incorporates capabilities to multiply, add, or subtract two polynomials from each other. It can also evaluate the Equation's value for any variable input, and can evaluate the results of raising the equation to any integer power.

b) ***Complex Number***

The solution of polynomial equations may contain combinations of real and complex numbers. The direct root solvers, (such as quadratic, cubic, and quartic equations) require extensive complex number mathematics. To handles these occurrences, it was necessary to create a robust Complex Number math function to allow the addition, subtraction, multiplication and power manipulation of complex numbers.

c) ***Formula***

This object is a combination of equations, which allows for very complex representations of mathematical systems.

d) ***Polynomial Derivative***

This is a method to calculate the first derivative of polynomial equations

e) ***Power Series***

Creates representations of Transcendental Functions (Sine, Cosine, and Exponential). This allows inclusion of transcendentals in the polynomial equation functions and root solvers.

f) ***General Power***

The Java Library function, Math.pow only calculates expressions that involve evaluating x^y. If x is zero, y must be greater than zero. If x is 0 or negative, y must be a whole number. The AdvancedMath function GeneralPower fills in all the gaps, and correctly calculates x^y for all cases of x and y.

g) ***RootSolver***

RootSolver utilizes the Quartic, Cubic, and Quadratic laws to solve 4th, 3rd, and 2nd order polynomial equations. It can rapidly, and correctly evaluate all real and complex roots for these types of equations.

h) ***NewtonsMethod***

NewtonsMethod is a more generic root solver. It can take any Polynomial Equation, or Formula containing Polynomial Equations, and rapidly solve for all real roots, returning them in ascending value. NewtonsMethod, as the name implies, employs Newton’s Method to identify a root. Once a root is located, the equation/formula is divided by this root to generate an equation of one lower power than previously solved. The process is repeated until no real roots remain (or until the equation becomes a 4th order equation, in which case RootSolver is called to directly solve for the roots).

3. Use of the AdvancedMath Package

The ASMD package (MoverSensorMediator) utilizes the AdvancedMath package to predict the precise times that the target will rise above or sink below the radar horizon of the sensor. It also uses this package to determine the times that the target enters and exits the range envelope of the sensor.

IV. ANALYSIS USING THE ASMD MODEL

In the previous chapters we have discussed the reasons for creating the ASMD Model, and briefly, how it works. In this section, we shall illustrate the analysis that can be conducted using the model.

The ASMD model was constructed to analyze two specific problems,

1) Determine the best screen arrangement for ships in a task force, and

2) Determine the best defensive firing policy.

Before we can analyze these issues, we will discuss the pertinent Measures of Effectiveness (MOE).

A. MEASURES OF EFFECTIVENESS

1. Alternatives

There are several alternatives for MOE selection. An obvious choice is to count the number (or percentage) of enemy missiles destroyed. Another is to evaluate the number (or percentage) of enemy missile hits against the HVU. Still a third option is the number (or percentage) of enemy missiles that achieve homing against the HVU.

2. Analysis of Alternatives

Once a raid has been launched, Missile Defense is best characterized as a defensive battle. As is the case in most defensive battles, the minimization of losses (either in equipment or territory) is the most important consideration. In view of this, we can discard a measurement of enemy missile destruction as a meaningful MOE for this model. The remaining two MOE alternatives require more discussion before a final selection can be made.

a) Number of Hits against the HVU.

Counting the Number of Hits against the HVU is, admittedly, a strong option as a defensive MOE. It quantifies the result that we most want to prevent, damage to the HVU. In order for this to be the best choice, however, we must have a very good measure of all of the properties that result in a hit. The model must properly simulate the entire flight path from time of launch, thru detection of target, counter-detection by and counter-attack by the target, and avoidance of terminal defenses before hitting the target.

b) Number of Missiles Homing on the HVU

If we consider only the number of missiles that achieve homing on the HVU, we do not need to consider the terminal defenses of the target ships. Homing will have been achieved at a distance substantially beyond terminal defense range. An advantage of this MOE, is that the process by which homing is established is largely a function of geometry. The motion of the missile can be precisely simulated and the conditions under which homing will be achieved (and against what ship) are very well understood. The terminal defense process is well understood, especially if a single missile is attacking the ship. What happens when there are more than one missile attacking the ship simultaneously is not particularly well defined. A final advantage of this MOE, is that is can adequately serve as a surrogate for measuring the hits against the HVU.

3. Selecting the MOE

Taking all of the above into consideration, it would appear that the most appropriate MOE for evaluating screen defenses (using the ASMD model) is quantifying the number of enemy missiles that achieve homing against the HVU. Considerable play-testing with the model has also revealed the fact that the HVU is extremely difficult to hit under any circumstances, and there is always a high degree of variance in the measure of hits, regardless of how many runs are made. The count of homing missiles shows far less variance, and does appear to be affected greatly by the defensive tactics utilized. Before proceeding further, let us state the obvious: if a missile does not achieve homing against the HVU, it cannot hit the HVU.

B. PRELIMINARY ANALYSIS

Preliminary analysis has focussed on the positioning of screening vessels relative to the HVU and the likely direction of the threat. It is important to recognize that none of the results presented here is final. This is due to three factors:

1. Classification of Data.

The exact dimensions of ships and performance of radar systems are classified and thereby excluded from this unclassified thesis.

2. Extremely Large and Diverse Design Variety

The ASMD model is capable of simulating an infinite variety of screen designs, threat axes and raid sizes. To definitively state that a "final" best solution had been found would require far more testing than was possible in the time frame of this thesis.

3. Wide Variance in Statistical Results

There is a significant relationship between the dispersion of incoming missile aimpoints and the distribution of homing missiles against a task force.

4. Suitability for Gaining Meaningful Insights

These limitations do not mean that we cannot conduct meaningful analysis using the ASMD model, however, the use of unclassified sources, operational experience, and reasonable assumptions can allow analysts to gain significant insights into proper screen design.

C. CLOSE DEFENSE OF A HVU USING 3 ESCORTS.

A small screen was constructed using an Aircraft Carrier, one Aegis Cruiser, and two Spruance class destroyers. The Cruiser was stationed ahead of the Carrier, and the destroyers were stationed on the flanks of the Carrier. The ranges of these stations were varied in order to identify the best distance to station the ships to counter a threat that could be mounted from any direction. Figure 10 illustrates this arrangement:

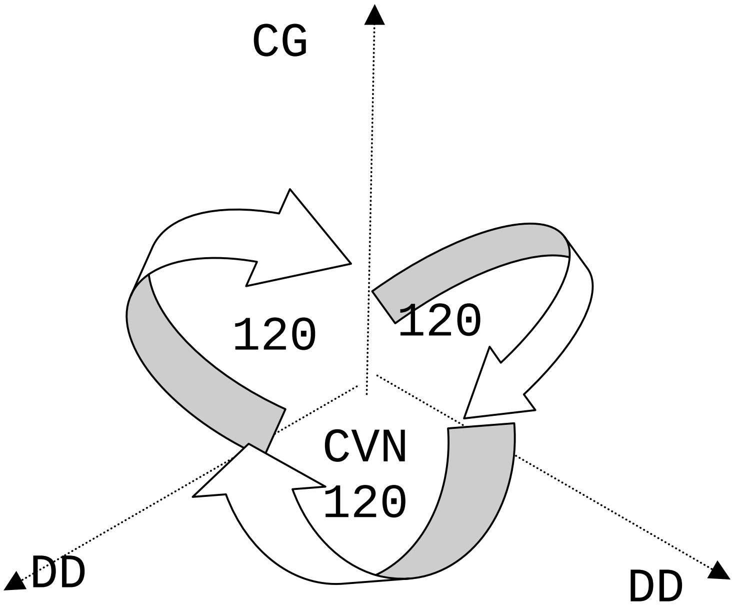


Figure 10 Screen Geometry

As can be seen, the screen is symmetrically distributed by azimuth. The simulated threat was a simultaneous raid by 10 Silkworm missiles. Raids were launched from 000R to 355R at 5 degree increments, and the raids were repeated 5 times in order to dampen the variance. The escorts are stationed 6 nautical miles (nm) from the carrier. Figure 11 illustrates the distribution of homing missiles against the screen. The charts on the following few pages were specifically built to show the threat posed to individual ships in the screen. An explanation of their interpretation and their method of construction is given in Appendix A.

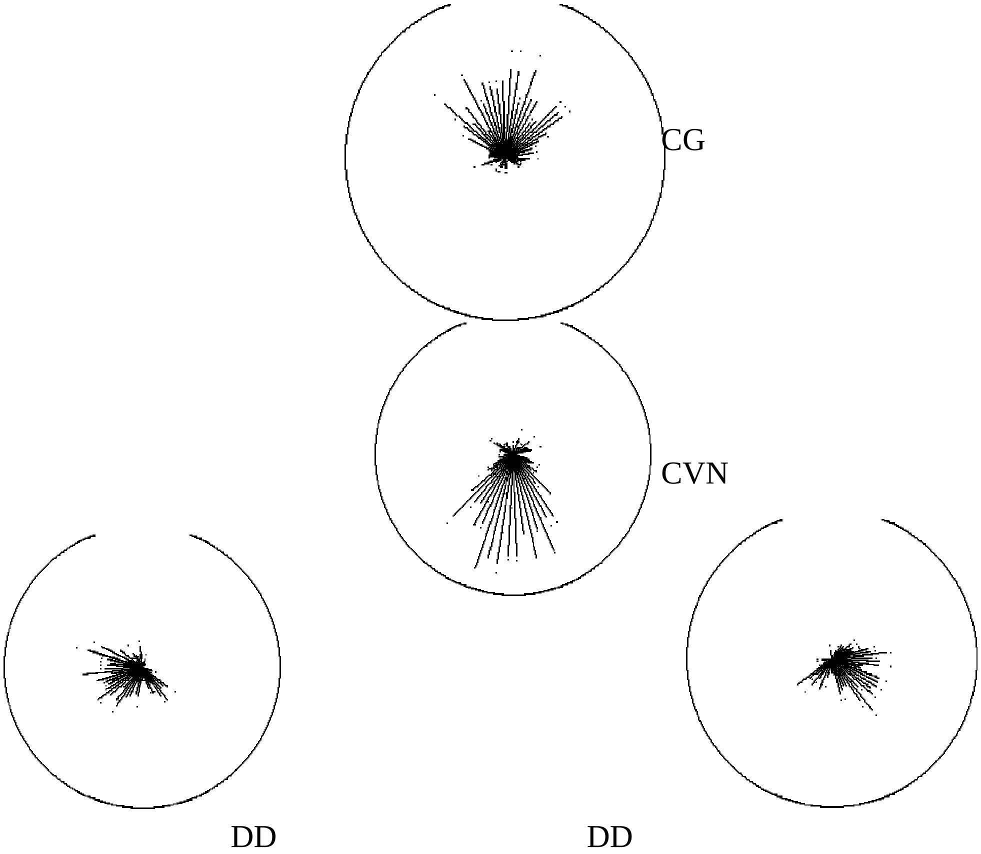
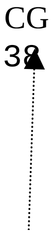


Figure 11 Distribution of Homing Missiles Against Screen

Of particular concern is the homing against the HVU, in this case the CVN-68 at the center of the screen. The next figure looks specifically at this distribution



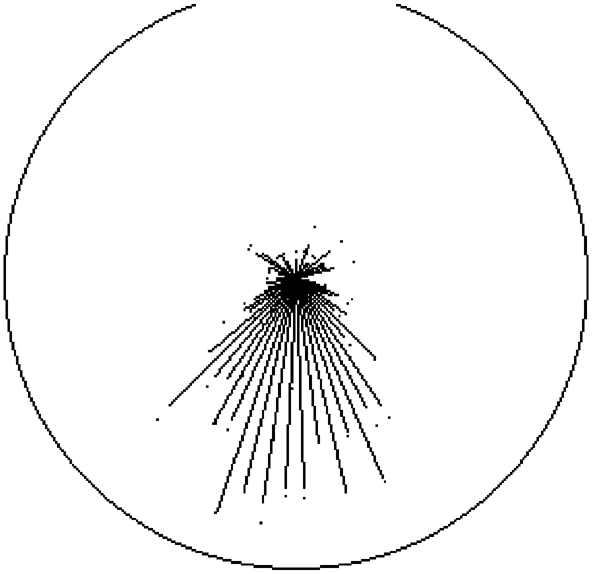
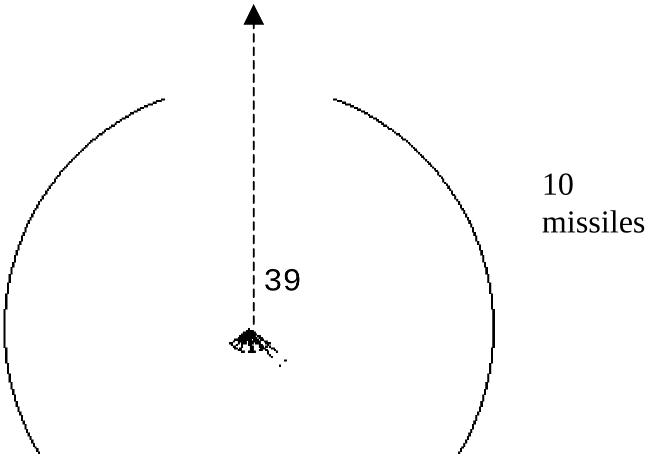


Figure 12 Homing on the HVU

It is easy to see that there is a large gap in protection afforded by this screen. Attacks originating from the area between the Destroyers and behind the Carrier achieve a high degree of success in homing against the Carrier.

In an effort to reduce this gap, let's first try moving the destroyers farther back, to 130 R and 230 R (vice 120 and 240 as before).

Now, the vulnerability to attacks from behind looks like this:



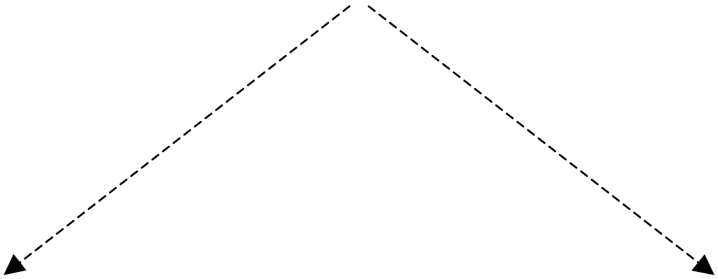


Figure 13 Homing with Destroyers moved closer together

A dramatic improvement was achieved by simply moving the destroyers a mere 10 degrees farther aft relative to the carrier.

Other analyses could be performed using the ASMD model to evaluate whether other combinations of screen spacing would be more or less effective in protecting the HVU.

D. DEFENSIVE FIRING POLICY

The second question that we examined was whether an SSL or SLS firing policy would be more effective (and under what conditions) in protecting the HVU. The defensive screen presented in Figure 10, with escorts stationed 6 nm from the CVN was exposed to missile attacks. The simulated threat was 10 Silkworm missiles. Separate analysis were conducted with attacks originating from a bearing of 000R and 060R. Fifty attacks from each bearing were simulated in order to reduce variance. Figures 14 and 15 summarize the Hit and Homing results from these attacks.

The charts clearly illustrate the superiority of SSL compared to SLS as a defensive firing policy against small numbers of missiles (in this case 10). Additional analysis has demonstrated that SSL remains superior until the number of missiles in the attack approaches 50 (if the screen contains a single cruiser). Above this number, the capacity of the Cruiser missile battery which initially contains 122 missiles, becomes a limiting item, and SLS starts to achieve parity. It should be noted that aside from our NATO allies, specifically Great Britain, and Russia, no country currently possesses the capability of mounting this large of an attack against our ships. This will probably not remain the case, however. Many nations, especially China, Iraq, Iran, and North Korea, are aggressively increasing their ASM inventories. ASM's present a cheap alternative to the construction of large navies, and can more rapidly 'level the playing field' than an expensive naval construction program. As a result, the problem of countering large scale ASM attacks will require considerable thought and analysis in the future.

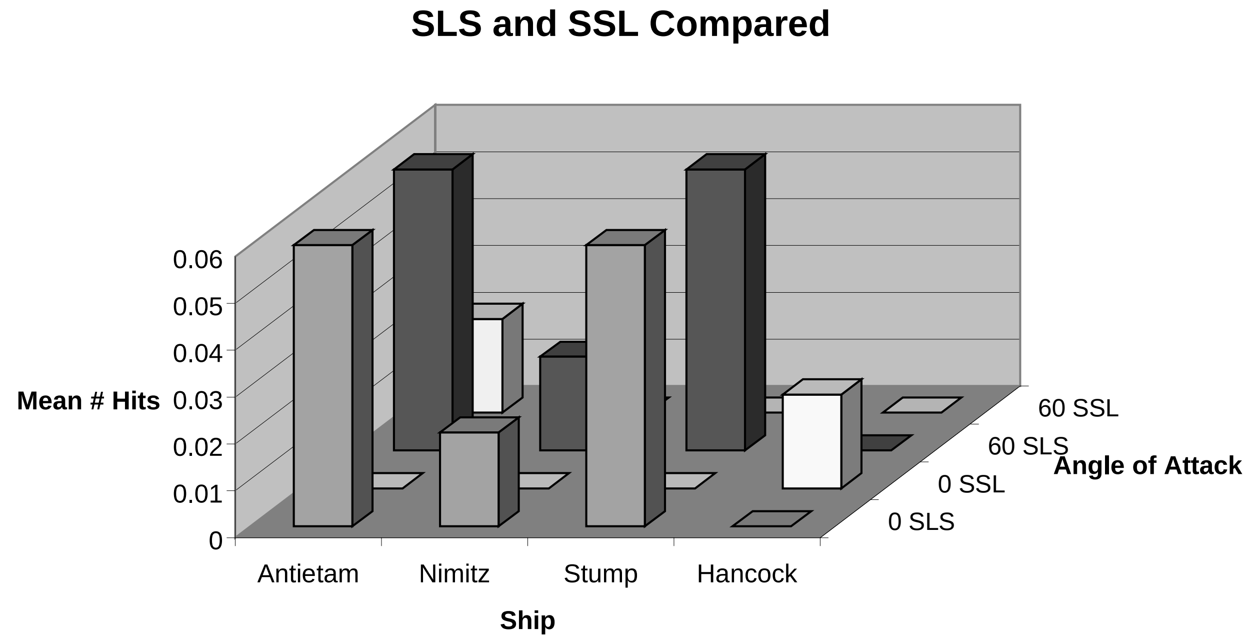


Figure 14 Hit Results SLS vs SSL

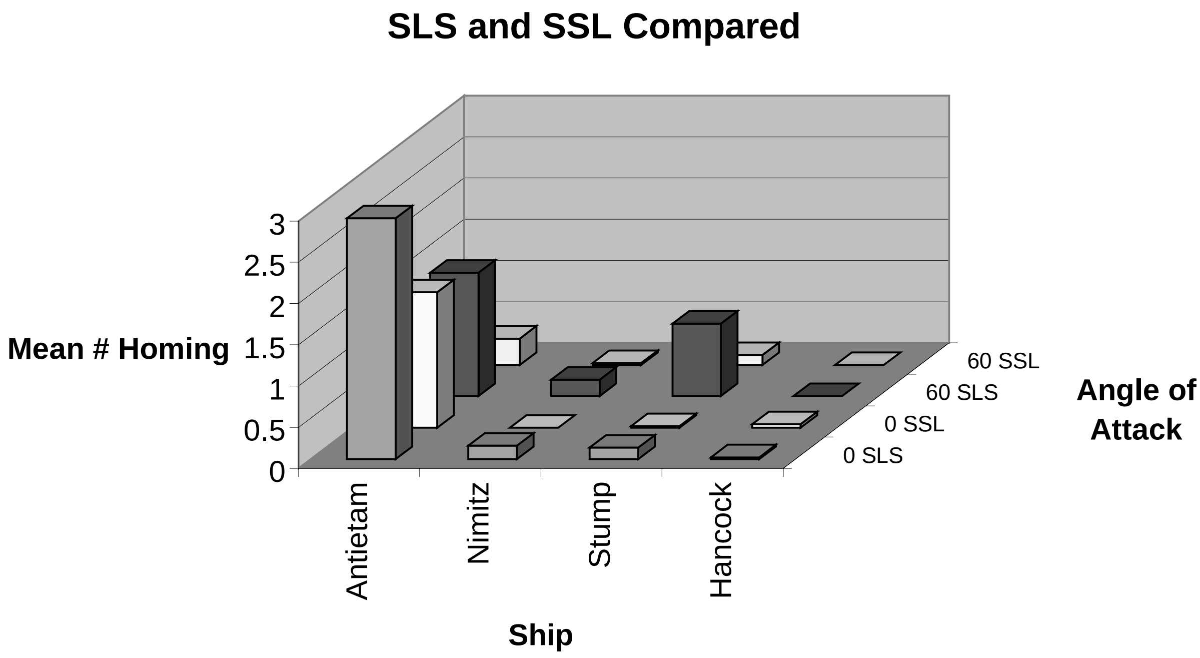


Figure 15 Homing Results SLS vs SSL

V CONCLUSIONS AND FUTURE WORK

A. The need for analysis

In this thesis, we have demonstrated the need for continued analysis in the area of Screen Defense against Anti-Ship Missile attacks. The Anti-Ship Missile Defense (ASMD) model has yielded significant insights into defense against small scale missile attacks, but further analysis is required because of the increasing threat posed by weapon proliferation, weapon capabilities, and the continued forward deployment of our warships.

B. Development of the ASMD Model

Existing simulation and model technology was found to be insufficient to conduct the analysis desired, so a new computer model (ASMD) was created. This new model more fully emulates the actual motion of missiles in space, with objects that accelerate and turn smoothly. The ASMD model features, as an adjunct, a sophisticated and highly capable mathematics package that can be utilized by future analysts to render moving objects even more accurately

We have sought to provide a method for analyzing the effectiveness of various defensive options, screen geometry, and defensive firing policy in enhancing the safety of the High Value Unit. Preliminary analysis was conducted using the ASMD model to illustrate the strengths and weaknesses of a few screen design and firing policy options.

C. Review of Preliminary Analysis using the ASMD model

Analysis using the ASMD model has borne out the superiority of Shoot-Shoot-Look defensive firing policies over Shoot-Look-Shoot in defending the ships against small to medium sized missile attacks (10 – 50 missiles). It has also been utilized to show the ability of proper escort placement to prevent enemy missiles from achieving homing against the HVU.

D. The need for further study

This study must be regarded as only the first step toward enhancing the safety of our naval ships at sea. Additional analysis, using more accurate and classified data will be necessary in order to state definitively the 'best' screen arrangements to counter a specific threat.

E. Other uses for the ASMD Model

While the ASMD model was developed with screen defense in mind, it may also be used to analyze many other situations. For example, it is well suited to the planning and execution of missile attacks against enemy warship formations. It can be utilized to evaluate the threat of multi-axis attacks and large scale missile raids as well as to evaluate the potential impact of decoy or other tactics in Naval Missile warfare.

LIST OF REFERENCES

Arntzen, A., *Software Components for Air Defense Planning*, Master's Thesis, Naval Postgraduate School Monterey, CA, 1998.

Bradley, G.H., Buss A.H., *An Architecture for Dynamic Planning Systems Using Loosely Coupled Components*, Proposal for Reimbursable Research, Naval Postgraduate School Monterey, CA, 1997.

Buss, A.H., Stork, K., "SIMKIT User's Manual", Naval Postgraduate School Monterey, CA, 1998.

Gamma, E., et al., *Design Patterns, Elements of Reusable Object-Oriented Software,* Addison-Wesley 1995.

Hughes, W.P., *Fleet Tactics: Theory and Practice*, Naval Institute Press, 1986.

Korzeniewski, G., Truelove, M.R., et al., "Simulation Technologies in the Joint Warfighting Environment", Science Applications International Corporation, McLean, VA, 1998.

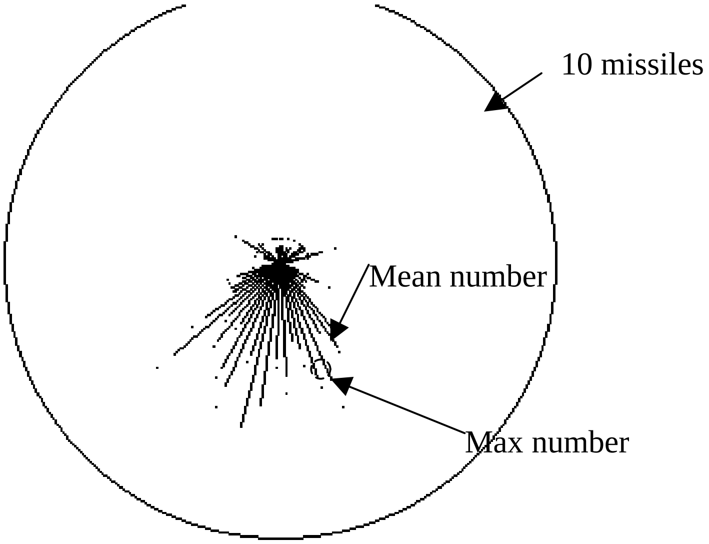
Pike, J., Bolkcom, C., “Attack Aircraft Proliferation: Issues for Concern”, Chapter 5, Federation of American Scientists, 1996.

Prezelin, B., Baker, A.D., *Combat Fleets of the World 1993, Their Ships, Aircraft, and Armament*, Naval Institute Press, 1993.

Stork, K., “Sensors in Object Oriented Discrete Event Simulation”, Master’s Thesis, Naval Postgraduate School Monterey, CA, 1996.

APPENDIX A. CREATION OF THREAT GRAPHS

During the creation and execution of the ASMD model, it was desired to create a highly effective and appealing chart that could summarize the threat posed to ships by attacks from various directions. The picture type illustrated below, is what was desired for this purpose.



Homing against the HVU

Unfortunately, none of the readily available software products available (Microsoft Excel, Microsoft Powerpoint, Corel Draw, Mathcad, and SPLUS) are capable of generating this type of graph. Because of this, it was necessary to build a computer program that would do the job.

The ASMD model, in its current incarnation, provides output data in the form of a text file. A Visual Basic program was created that could read this text output and convert it into a Bitmap format picture. This picture can then be imported into any standard graphics capable program for manipulation and display. The computer code necessary to accomplish this conversion is provided below. It should be noted that this code was written to accommodate the specific text format generated by the ASMD model. It could be modified to handle other formats, if desired. Future versions of the ASMD model will feature an organic graphing capability written in the Java programming language. The time constraints of the thesis precluded this from being developed earlier.

1. Summary of Program

In a nutshell, the user is prompted to select a text file for conversion. Once he makes a selection, the program opens the desired file and analyzes it for the following pertinent data:

-Number and Names of Ships

-Number and value of attack angles

-Number and value of missile raid sizes

These are used in order to create internal data arrays that will be filled by the program.

After this preliminary search is accomplished, the program automatically scans the text file for information regarding the number of hits, number of missiles that achieve homing, and the number of defensive missiles fired by each ship at raids originating from each attack angle and for each missile raid size simulated. This information is contained in text fields that provide values of Mean, Maximum, Minimum, and Standard Deviation for the trial that was conducted.

A separate Visual Basic Form (essentially a display window) is created for each combination of Ship and Missile Raid Size. Lines are drawn on each form that correspond to the mean value of hits, homing, and shots fired for each angle of attack.

Finally, the program saves the attack graph that was created on each form as a Bitmap formatted picture with a unique name that identifies the ship and the missile raid size.

2. Visual Basic Source Code

```
Private Sub cmdStart_Click()
   Dim fileName As String
   dial1.ShowOpen
   fileName = dial1.fileName
   ReadFile (fileName)
End Sub

Private Sub ReadFile(fileName As String)
   Open fileName For Input As #1
   Dim ShipName() As String
   ReDim ShipName(10)
   Dim longString As String
   Dim msg As String
   Dim enabled As Boolean
   enabled = True
   Dim nameEnabled As Boolean
   nameEnabled = True
   Dim testCharacter As String
   Dim ship As Integer
   Dim chrPos As Integer
   'Find the names of ships

   'Now look for raid sizes and angular data
   enabled = True
   Dim raid() As Integer 'raid size
   ReDim raid(20)
   Dim raidNumber As Integer
   raidNumber = 0
   Dim angle() As Double
   ReDim angle(360) 'allow 1 degree spacing, will later truncate
   Dim angleNumber As Integer
   angleNumber = 0
'   Open fileName For Input As #1
   Do While Not EOF(1)
      Line Input #1, longString
      If UCase$(Left$(longString, 4)) = "RAID" Then
         chrPos = 1
         testCharacter = "T"
         While testCharacter <> Chr$(13) And testCharacter <> ""
            testCharacter = Mid$(longString, chrPos, 1)
            If testCharacter > Chr$(47) And testCharacter < Chr$(58) Then 'a number
               msg = msg + testCharacter
            End If
            chrPos = chrPos + 1
         Wend
         raid(raidNumber) = Val(msg)
         raidNumber = raidNumber + 1
         If raidNumber > 1 Then enabled = False
         msg = ""
      End If
      If Left$(longString, 4) = "Name" And nameEnabled Then ’start line
         While Left$(longString, 5) <> "Angle"
            Line Input #1, longString
            chrPos = 1
            Do While testCharacter <> Chr$(9) And chrPos < 35
               testCharacter = Mid$(longString, chrPos, 1)
               ShipName(ship) = ShipName(ship) + testCharacter
               chrPos = chrPos + 1
            Loop
            ShipName(ship) = Left$(ShipName(ship), chrPos - 2)
            ship = ship + 1
            testCharacter = "T"
         Wend
         ReDim Preserve ShipName(ship - 2)
         ship = ship - 1
         nameEnabled = False
      End If
      If UCase$(Left$(longString, 5)) = "ANGLE" And enabled Then
         chrPos = 1
         testCharacter = "T"
         While testCharacter <> Chr$(13) And testCharacter <> ""
            testCharacter = Mid$(longString, chrPos, 1)
            If testCharacter > Chr$(47) And testCharacter < Chr$(58) Then ’a number
               msg = msg + testCharacter
            End If
            If testCharacter = Chr$(46) Then msg = msg + testCharacter ’decimal
            chrPos = chrPos + 1
         Wend
         angle(angleNumber) = Val(msg)
         angleNumber = angleNumber + 1
         msg = ""
      End If
   Loop
   ReDim Preserve angle(angleNumber - 1)
   ReDim Preserve raid(raidNumber - 1)
   Close #1
   Dim shipNumber As Integer
   Dim shipForm() As New frmShip
   Dim numberForms As Integer
   numberForms = (raidNumber) * (ship)
   ReDim shipForm(numberForms)
   Dim formNumber As Integer
   formNumber = 0
   Dim nameOfForm As String
```

```
    For numberRaid = 0 To raidNumber - 1
        For shipNumber = 0 To ship - 1
            nameOfForm = Left$(fileName, Len(fileName) - 4) + ShipName(shipNumber) + "Raid size" +
Str(raid(numberRaid))
            With shipForm(formNumber)
                .Initialize (nameOfForm)
                .Width = MDIForm1.Height / 3#
                .Height = .Width
                .Visible = True
                .Left = shipNumber * .Width
                .Top = numberRaid * .Height
            End With
            formNumber = formNumber + 1
        Next shipNumber
    Next numberRaid
    shipNumber = ship
    ’Now work on getting the actual data
    Open fileName For Input As #1
    Dim numberHits() As Double
    Dim standardDevHits() As Double
    Dim maxHits() As Integer
    Dim minHits() As Integer
    Dim numberHome() As Double
    Dim standardDevHome() As Double
    Dim maxHome() As Integer
    Dim minHome() As Integer
    Dim numberShots() As Double
    Dim standardDevShots() As Double
    Dim maxShots() As Integer
    Dim minShots() As Integer
    ReDim numberHits(raidNumber, angleNumber, shipNumber)
    ReDim standardDevHits(raidNumber, angleNumber, shipNumber)
    ReDim maxHits(raidNumber, angleNumber, shipNumber)
    ReDim minHits(raidNumber, angleNumber, shipNumber)
    ReDim numberHome(raidNumber, angleNumber, shipNumber)
    ReDim standardDevHome(raidNumber, angleNumber, shipNumber)
    ReDim maxHome(raidNumber, angleNumber, shipNumber)
    ReDim minHome(raidNumber, angleNumber, shipNumber)
    ReDim numberShots(raidNumber, angleNumber, shipNumber)
    ReDim standardDevShots(raidNumber, angleNumber, shipNumber)
    ReDim maxShots(raidNumber, angleNumber, shipNumber)
    ReDim minShots(raidNumber, angleNumber, shipNumber)
’   Dim numberRaid As Integer
    Dim numberAngle As Integer
    Dim numberShip As Integer
    Dim testString As String
    Do While Not EOF(1)
        For numberRaid = 0 To raidNumber - 1
            For numberAngle = 0 To angleNumber - 1
                While Left$(testString, 4) <> "Name" And Not EOF(1)
                    Line Input #1, testString
                Wend
            Dim dataBit As Integer
            dataBit = 1
            For numberShip = 0 To shipNumber - 1
               'Find first ship data
               If EOF(1) Then Exit Do
               Line Input #1, testString
               chrPos = Len(ShipName(numberShip)) + 1
reTry:
               If Mid$(testString, chrPos, 1) = Chr$(9) Then
                  chrPos = chrPos + 1
                  GoTo reTry
               End If
               For dataBit = 1 To 12
                  msg = ""
                  Do While Mid$(testString, chrPos, 1) <> Chr$(9)
                     testCharacter = Mid$(testString, chrPos, 1)
                     msg = msg + testCharacter
                     chrPos = chrPos + 1
                  Loop
                  If Right$(msg, 1) = Chr$(9) Then msg = Left$(msg, Len(msg) - 1)
                  Select Case dataBit
                  Case 1:
                     numberHits(numberRaid, numberAngle, numberShip) = Val(msg)
                  Case 2:
                     standardDevHits(numberRaid, numberAngle, numberShip) = Val(msg)
                  Case 3:
                     maxHits(numberRaid, numberAngle, numberShip) = Int(Val(msg))
                  Case 4:
                     minHits(numberRaid, numberAngle, numberShip) = Int(Val(msg))
                  Case 5:
                     numberHome(numberRaid, numberAngle, numberShip) = Val(msg)
                  Case 6:
                     standardDevHome(numberRaid, numberAngle, numberShip) = Val(msg)
                  Case 7:
                     maxHome(numberRaid, numberAngle, numberShip) = Int(Val(msg))
                  Case 8:
                     minHome(numberRaid, numberAngle, numberShip) = Int(Val(msg))
                  Case 9:
                     numberShots(numberRaid, numberAngle, numberShip) = Val(msg)
                  Case 10:
                     standardDevShots(numberRaid, numberAngle, numberShip) = Val(msg)
                  Case 11:
                     maxShots(numberRaid, numberAngle, numberShip) = Int(Val(msg))
                  Case 12:
                     minShots(numberRaid, numberAngle, numberShip) = Int(Val(msg))
                  End Select
                  chrPos = chrPos + 1
               Next dataBit
              Next numberShip
         Next numberAngle
      Next numberRaid
   Loop
```

```
  Close #1
  ’draw the lines
  formNumber = 0
  For numberRaid = 0 To raidNumber - 1
    For numberShip = 0 To shipNumber - 1
      For numberAngle = 0 To angleNumber - 1
        Dim angleRadians As Double
        angleRadians = angle(numberAngle) * 3.14159 / 180
        Dim x1 As Single
        Dim x2 As Single
        Dim y1 As Single
        Dim y2 As Single
        x1 = 0
        y1 = 0
        x2 = 0
        y2 = 0
        Dim deltaX As Double
        Dim deltaY As Double
        Dim minValue As Double
        Dim maxValue As Double
    ’Determine standard deviation spread and plot it
        minValue = numberHits(numberRaid, numberAngle, numberShip) -
standardDevHits(numberRaid, numberAngle, numberShip)
        If minValue < 0 Then minValue = 0
              minValue = numberHits(numberRaid, numberAngle, numberShip) +
              standardDevHits(numberRaid, numberAngle, numberShip)
        deltaX = Sin(angleRadians) * minValue
        deltaY = Cos(angleRadians) * minValue
        x1 = deltaX
        y1 = deltaY
        x2 = Sin(angleRadians) * maxValue
        y2 = Cos(angleRadians) * maxValue
        shipForm(formNumber).Line (x1, y1)-(x2, y2), QBColor(4)
        x2 = Sin(angleRadians) * maxHits(numberRaid, numberAngle, numberShip)
        y2 = Cos(angleRadians) * maxHits(numberRaid, numberAngle, numberShip)
        DrawWidth = 5
        shipForm(formNumber).PSet (x2, y2), QBColor(4)
        x2 = Sin(angleRadians) * minHits(numberRaid, numberAngle, numberShip)
        y2 = Cos(angleRadians) * minHits(numberRaid, numberAngle, numberShip)
        shipForm(formNumber).PSet (x2, y2), QBColor(4)
        DrawWidth = 1
        angleRadians = angleRadians + (3 * 3.14159 / 180)
        minValue = numberHome(numberRaid, numberAngle, numberShip) -
standardDevHome(numberRaid, numberAngle, numberShip)
        If minValue < 0 Then minValue = 0
        minValue = numberHome(numberRaid, numberAngle, numberShip) +
standardDevHome(numberRaid, numberAngle, numberShip)
        deltaX = Sin(angleRadians) * minValue
        deltaY = Cos(angleRadians) * minValue
        x1 = deltaX
        y1 = deltaY
        x2 = Sin(angleRadians) * maxValue
```

```
                y2 = Cos(angleRadians) * maxValue
                shipForm(formNumber).Line (x1, y1)-(x2, y2), QBColor(5)
                x2 = Sin(angleRadians) * maxHome(numberRaid, numberAngle, numberShip)
                y2 = Cos(angleRadians) * maxHome(numberRaid, numberAngle, numberShip)
                DrawWidth = 5
                shipForm(formNumber).PSet (x2, y2), QBColor(5)
                x2 = Sin(angleRadians) * minHome(numberRaid, numberAngle, numberShip)
                y2 = Cos(angleRadians) * minHome(numberRaid, numberAngle, numberShip)
                shipForm(formNumber).PSet (x2, y2), QBColor(5)
                DrawWidth = 1
 ’              angleRadians = angleRadians - (6 * 3.14159 / 180)
 ’              minValue = numberShots(numberRaid, numberAngle, numberShip) -
standardDevShots(numberRaid, numberAngle, numberShip)
 ’              If minValue < 0 Then minValue = 0
 ’              minValue = numberShots(numberRaid, numberAngle, numberShip) +
standardDevShots(numberRaid, numberAngle, numberShip)
 ’              deltaX = Sin(angleRadians) * minValue
 ’              deltaY = Cos(angleRadians) * minValue
 ’              x1 = deltaX
 ’              y1 = deltaY
 ’              x2 = Sin(angleRadians) * maxValue
 ’              y2 = Cos(angleRadians) * maxValue
 ’              shipForm(formNumber).Line (x1, y1)-(x2, y2), QBColor(1)
 ’              x2 = Sin(angleRadians) * maxShots(numberRaid, numberAngle, numberShip)
 ’              y2 = Cos(angleRadians) * maxShots(numberRaid, numberAngle, numberShip)
 ’              DrawWidth = 5
 ’              shipForm(formNumber).PSet (x2, y2), QBColor(1)
 ’              x2 = Sin(angleRadians) * minShots(numberRaid, numberAngle, numberShip)
 ’              y2 = Cos(angleRadians) * minShots(numberRaid, numberAngle, numberShip)
 ’              shipForm(formNumber).PSet (x2, y2), QBColor(1)
                DrawWidth = 1
            Next numberAngle
            formNumber = formNumber + 1
        Next numberShip
    Next numberRaid
    For formNumber = 0 To numberForms - 1
        For raidSize = 10 To 40 Step 10
            shipForm(formNumber).Circle (0, 0), raidSize
        Next raidSize
    Next formNumber
End Sub
```

APPENDIX B: JAVA SOURCE CODE FOR THE MASTER ASMD PROGRAM

The ASMD model is a large and complex computer program. Future analysts should not be concerned, too much, with the intricacies of the entire program, however. All of the individual components that make up the model can readily be utilized by a relatively small and simple master program. The operation of this master program will be summarized in this annex. The Java source code is provided (and annotated) to illustrate the ease with which screens may be designed and analyzed. The ASMD model itself is open-source, meaning that any person who desires to utilize it may do so by obtaining a copy from the Operations Research Department of the Naval Postgraduate School.

A. SUMMARY OF PROGRAM OPERATION

The master ASMD Program, currently called TestRegistry, is simple but powerful. We will analyze the major functional blocks, and then provide the source code in its entirety.

1. Preliminary Material

This section of code contains the computer calls to outside libraries that are needed for program execution.

```
package ASMD;
import modkit.*;˝
//import Working.*
import simkit.*;
import modutil.spatial.*;
import modsim.*;
import java.util.*;
import simkit.data.*;
import java.text.NumberFormat;
import java.util.Locale;
```

2. Setting up the Scenario

In this section, TestRegistry creates the Anti-Ship Missile Attack Scenario that will be analyzed.

```
public class TestRegistry extends BasicModSimComponent {

public static void main(String[] args) {
 Locale loc = Locale.US;
 NumberFormat nf = NumberFormat.getInstance(loc);
```

Missile raids will be conducted, starting with 10 missiles in the raid:

```
 int numberOfMissiles = 10;
```

There will be 4 ships in the screen:

```
 BasicModSimComponent[] ship = new BasicModSimComponent[4];
```

The program will terminate with a maximum raid size of 10 missiles:

```
 while (numberOfMissiles <= 10) {
  System.out.println("Raid Size = " + numberOfMissiles);
```

We will examine missile attacks from relative bearings between 120R and 240R:

```
  double angle = 120.0;
  while (angle < 240) {
```

This next section sets up the data collection parameters for the attack:

```
   SimpleStats[] hitCounter = new SimpleStats[5];
   SimpleStats[] homesCounter = new SimpleStats[5];
   SimpleStats[] shotsCounter = new SimpleStats[5];
   SimpleStats missilesShotDown = new SimpleStats(SamplingType.TALLY);
   SimpleStats missilesPassive = new SimpleStats(SamplingType.TALLY);
   SimpleStats missilesFuel = new SimpleStats(SamplingType.TALLY);
   for (int i = 0; i < 4; i++) {
    hitCounter[i] = new SimpleStats(SamplingType.TALLY);
    homesCounter[i] = new SimpleStats(SamplingType.TALLY);
    shotsCounter[i] = new SimpleStats(SamplingType.TALLY);
   }
```

A total of 5 runs will be conducted at each attack angle (and for each missile raid size):

```
   for (int runCount = 1; runCount <= 5; runCount++) {
```

Creates a data collection entity:

```
Tabulator tab = new Tabulator();
```

Specifies that a Shoot-Look-Shoot (SLS) defensive firing policy will be in effect. One missile will be fired in self defense each time. SSL would be specified as (“SSL”, 2):

```
FireMode fireMode = new FireMode("SLS", 1);
```

Creates the Register utilizing two sides in this battle, BLUE and RED:

```
Side[] colors = {Side.BLUE, Side.RED};
NewRegister r = new NewRegister(colors, tab, fireMode);
```

Creates individual ships from pre-defined ship classes. These ship classes feature all of the necessary movement, sensor, and weapon data to conduct the analysis. Each ship is assigned to the BLUE side:

```
CG47 Antietam = new CG47("Antietam", Side.BLUE);
CVN Nimitz = new CVN("Nimitz", Side.BLUE);
DD963 Stump = new DD963("Stump", Side.BLUE);
DD963 Moos = new DD963("Hancock", Side.BLUE);
```

Gives each of the ships a unique name for identification purposes:

```
ship[0] = Antietam;
ship[1] = Nimitz;
ship[2] = Stump;
ship[3] = Moos;
```

The next section creates a screen with Nimitz as the guide. It defines screen sectors and assign ships to them:

between 130R and 150R, 2 nm to 4 nm for Stump
between 210R and 230R, 2 nm to 4 nm for Moos
between -π/8 (-045R) and π/8 (045R), 4 nm to 8 nm for Antietam

```
for (int shipNumber = 0; shipNumber < ship.length; shipNumber++) {
        r.addUnit(ship[shipNumber], true);
}
Screen screen = new Screen("Blue Screen");
 screen.addGuide(new ScreenLocation(new Coor4D(0, 0, 230, 0), Nimitz));
 double angle1 = 130.0 * Math.PI/180.0;
 double angle2 = 150.0 * Math.PI/180.0;
 screen.addUnit(new ScreenLocation(new Coor4D(2, 4, angle1, angle2), Stump));
 angle1 = 210.0 * Math.PI/180.0;
 angle2 = 230.0 * Math.PI/180.0;
 screen.addUnit(new ScreenLocation(new Coor4D(2, 4, angle1, angle2), Moos));
 screen.addUnit(new ScreenLocation(new Coor4D(4, 8, -Math.PI/8.0, Math.PI/8.0), Antietam));
```

The next 2 lines create a track for the screen. The guide will pass thru location (18.0, 47.0) at 4 hours into the problem, and

(36.0, 48.2) at 7 hours.
The 3600 factor is needed to convert the time into seconds.

```
screen.setScreenDestination(new Coor4D(18.0, 47.0, 0, 4*3600));
screen.setScreenDestination(new Coor4D(36.0, 48.2, 0, 7*3600));
```

The next 4 lines created a new Silkworm missile site. It is located 35 miles from the center of the screen, and it's angle of attack can be varied by iterating thru the values of <u>angle</u> desired. It provides a total of 200 missiles to the site, and assigns it to the RED side.

```
double silkX = 35 * Math.sin(Math.PI * angle/180.0);
double silkY = 35 * Math.cos(Math.PI * angle/180.0);
SilkwormSite site1 = new SilkwormSite(new Coor3D(silkX, silkY, 0), 200, Side.RED);
r.addSite(site1);
```

These 3 lines of code set up the target area for the silkworm site. It will conduct an attack against an elliptical AOU (center at (0, 0), oriented 000T, with semi-major axis 35 miles, and semi-minor axis 25 miles). The attack will consist of 10 missiles and will start at time 60 seconds.

```
Ellipse aou = new Ellipse(new Coor2D(0, 0), 0.0, 35.0, 25.0);
FireMission f1 = new FireMission(60.0, numberOfMissiles, new Coor2D(0, 0), aou);
site1.setMission(f1);
```

a) Starting a Run

These 5 lines of code start the execution, and direct that the run will stop at 600 seconds of simulated time.

```
Schedule.clearRerun();
Schedule.setSingleStep(false);
Schedule.setVerbose(false);
Schedule.stopOnTime(600.0);
Schedule.startSimulation();
```

b) Collecting the data

The code below causes the program to collect data for each ship from the run. This data consists of the number of times that an individual ship was hit, the number of missiles that achieved homing against it, and the number of Surface to Air Missiles it fired in defense of the screen.

```
int hits;
int homes;
int shots;
for (int shipNumber = 0; shipNumber < ship.length; shipNumber++) {
  BasicModSimComponent thisShip = ship[shipNumber];
  hits =  ((Integer) thisShip.getProperty("Hits", new Integer(0))).intValue();
  homes = ((Integer) thisShip.getProperty("HomedBy", new Integer(0))).intValue();
  shots = ((Integer) thisShip.getProperty("MissileShots", new Integer(0))).intValue();
  hitCounter[shipNumber].newObservation(hits);
  homesCounter[shipNumber].newObservation(homes);
  shotsCounter[shipNumber].newObservation(shots);
}
```

The following 3 lines cause data to be recorded regarding missile performance. Specifically the program records the number of ASM that were shot down by SAMs, the number of ASM that were killed by point defenses or spoofed by chaff/EW, and the number of ASMs that ran out of fuel.

```
missilesShotDown.newObservation(tab.getMissilesKilledByMissiles());
missilesPassive.newObservation(tab.getMissilesKilledByPassive());
missilesFuel.newObservation(tab.getMissilesKilledByFuel());
```

c) Setting up for the next run

This code destroys the objects that were created for the run just completed. This frees up computer memory for the execution of another trial. This process is repeated until the specified number of trial runs at this attack angle and with this number of missiles (in this case, a total of 5 runs had been specified) have been completed.

```
  screen.reset();
  r.reset();
  site1 = null;
  site2 = null;
  site3 = null;
  r = null;
  screen = null;
  Schedule.reset();
}
```

d) Reporting the Data

This code section causes a formatted report of the statistical results of the 5 runs to be displayed. These results consists of the mean, standard deviation of hits, maximum, minimum numbers of hits, homings, and shots fired for each ship. In addition, these same statistics are displayed for the numbers of ASMs shot down, the number that were killed either passively or by point defense systems (grouped into a single 'Passive' category), and the numbers of ASMs that ran out of fuel.

```
System.out.println("Angle of attack " + angle);
System.out.println("Averages by ship");
System.out.println(" " + '\t' + '\t' + "HITS by Enemy Missiles" + '\t'
            + '\t' + "HOMING by Enemy Missiles" + '\t'
            + '\t' + "SHOTS at Enemy Missiles" + '\t'
```

```
                    + '\t’ + '\t’ + "Enemy missiles Killed"  );
System.out.println("Name" + '\t’ + '\t’
  + "Mean" + '\t’ + "Std Dev" + '\t’ + "Max" + '\t’ + "Min" + '\t’
  + "Mean" + '\t’ + "Std Dev" + '\t’ + "Max" + '\t’ + "Min" + '\t’
  + "Mean" + '\t’ + "Std Dev" + '\t’ + "Max" + '\t’ + "Min" + '\t’
  + '\t’ + "Mean" + '\t’ + "Std Dev" + '\t’ + "Max" + '\t’ + "Min");
for (int shipNumber = 0; shipNumber < ship.length; shipNumber++) {
  String name = ship[shipNumber].getName();
  String msg = name + '\t’ + '\t’;
  msg = msg + nf.format(hitCounter[shipNumber].getMean()) + '\t’;
  msg = msg + nf.format(hitCounter[shipNumber].getStandardDeviation()) + '\t’;
  msg = msg + nf.format(hitCounter[shipNumber].getMaxObs()) + '\t’;
  msg = msg + nf.format(hitCounter[shipNumber].getMinObs()) + '\t’;
  msg = msg + nf.format(homesCounter[shipNumber].getMean()) + '\t’;
  msg = msg + nf.format(homesCounter[shipNumber].getStandardDeviation()) + '\t’;
  msg = msg + nf.format(homesCounter[shipNumber].getMaxObs()) + '\t’;
  msg = msg + nf.format(homesCounter[shipNumber].getMinObs()) + '\t’;
  msg = msg + nf.format(shotsCounter[shipNumber].getMean()) + '\t’;
  msg = msg + nf.format(shotsCounter[shipNumber].getStandardDeviation()) + '\t’;
  msg = msg + nf.format(shotsCounter[shipNumber].getMaxObs()) + '\t’;
  msg = msg + nf.format(shotsCounter[shipNumber].getMinObs()) + '\t’;
  if (shipNumber == 1) {
    msg = msg + "Missile" + '\t’ ;
    msg = msg+ nf.format(missilesShotDown.getMean()) + '\t’;
    msg = msg + nf.format(missilesShotDown.getStandardDeviation()) + '\t’;
    msg = msg + nf.format(missilesShotDown.getMaxObs()) + '\t’;
    msg = msg + nf.format(missilesShotDown.getMinObs()) + '\t’;
  }
  if (shipNumber == 2) {
    msg = msg + "Passive" + '\t’;
    msg = msg+ nf.format(missilesPassive.getMean()) + '\t’;
    msg = msg + nf.format(missilesPassive.getStandardDeviation()) + '\t’;
    msg = msg + nf.format(missilesPassive.getMaxObs()) + '\t’;
    msg = msg + nf.format(missilesPassive.getMinObs()) + '\t’;
  }
  if (shipNumber == 3) {
    msg = msg + "Fuel" + '\t’;
    msg = msg+ nf.format(missilesFuel.getMean()) + '\t’;
    msg = msg + nf.format(missilesFuel.getStandardDeviation()) + '\t’;
    msg = msg + nf.format(missilesFuel.getMaxObs()) + '\t’;
    msg = msg + nf.format(missilesFuel.getMinObs()) + '\t’;
  }

  System.out.println(msg);
}
for (int aa = 0; aa < 4; aa++) {
  hitCounter[aa].reset();
  homesCounter[aa].reset();
  shotsCounter[aa].reset();
}
```

e) Analyzing other Attack Angles and Raid Sizes

Up to this point, we will have executed 5 runs that simulated the attack of 10 Silkworm missiles fired from 120R. The remaining code causes the angle of attack to increment by 5 degrees so that we examine the entire arc of fire, from 120R to 240R in 5 degree segments.

```
  angle = angle + 5;

    }
```

If we had specified a maximum number of missiles greater than 10, the following code would cause a repetition of all of the angular attacks for 20 missiles, then 30 missiles, and so on, until the maximum raid size had been achieved.

```
    numberOfMissiles = numberOfMissiles + 10;
   }
```

Finally, the remaining code terminates the program.

```
   System.exit(0);
  }
}
```

3. TestRegistry Code, in it's Entirety

As promised, the following is the entire Java source code for the TestRegistry program:

```
package ASMD;
import modkit.*;
import simkit.*;
import modutil.spatial.*;
import modsim.*;
import java.util.*;
import simkit.data.*;
import java.text.NumberFormat;
import java.util.Locale;

public class TestRegistry extends BasicModSimComponent {

  public static void main(String[] args) {
    Locale loc = Locale.US;
    NumberFormat nf = NumberFormat.getInstance(loc);
    int numberOfMissiles = 10;
    BasicModSimComponent[] ship = new BasicModSimComponent[4];
    while (numberOfMissiles <= 10) {
```

```
System.out.println("Raid Size = " + numberOfMissiles);
double angle = 120.0;
while (angle < 240) {
  SimpleStats[] hitCounter = new SimpleStats[5];
  SimpleStats[] homesCounter = new SimpleStats[5];
  SimpleStats[] shotsCounter = new SimpleStats[5];
  SimpleStats missilesShotDown = new SimpleStats(SamplingType.TALLY);
  SimpleStats missilesPassive = new SimpleStats(SamplingType.TALLY);
  SimpleStats missilesFuel = new SimpleStats(SamplingType.TALLY);
  for (int i = 0; i < 4; i++) {
    hitCounter[i] = new SimpleStats(SamplingType.TALLY);
    homesCounter[i] = new SimpleStats(SamplingType.TALLY);
    shotsCounter[i] = new SimpleStats(SamplingType.TALLY);
  }
  for (int runCount = 1; runCount <= 5; runCount++) {
    Tabulator tab = new Tabulator();
    FireMode fireMode = new FireMode("SLS", 1);
    Side[] colors = {Side.BLUE, Side.RED};
    NewRegister r = new NewRegister(colors, tab, fireMode);
    CG47 Antietam = new CG47("Antietam", Side.BLUE);
    CVN Nimitz = new CVN("Nimitz", Side.BLUE);
    DD963 Stump = new DD963("Stump", Side.BLUE);
    DD963 Moos = new DD963("Hancock", Side.BLUE);
    ship[0] = Antietam;
    ship[1] = Nimitz;
    ship[2] = Stump;
    ship[3] = Moos;
    for (int shipNumber = 0; shipNumber < ship.length; shipNumber++) {
      r.addUnit(ship[shipNumber], true);
    }
    Screen screen = new Screen("Blue Screen");
    screen.addGuide(new ScreenLocation(new Coor4D(0, 0, 230, 0), Nimitz));
    double angle1 = 130.0 * Math.PI/180.0;
    double angle2 = 150.0 * Math.PI/180.0;
    screen.addUnit(new ScreenLocation(new Coor4D(2, 4, angle1, angle2), Stump));
    angle1 = 210.0 * Math.PI/180.0;
    angle2 = 230.0 * Math.PI/180.0;
    screen.addUnit(new ScreenLocation(new Coor4D(2, 4, angle1, angle2), Moos));
    screen.addUnit(new ScreenLocation(new Coor4D(4, 8, -Math.PI/8.0, Math.PI/8.0), Antietam));
    screen.setScreenDestination(new Coor4D(18.0, 47.0, 0, 4*3600));
    screen.setScreenDestination(new Coor4D(36.0, 48.2, 0, 7*3600));
    double silkX = 35 * Math.sin(Math.PI * angle/180.0);
    double silkY = 35 * Math.cos(Math.PI * angle/180.0);
    SilkwormSite site1 = new SilkwormSite(new Coor3D(silkX, silkY, 0), 200, Side.RED);
    SilkwormSite site2 = new SilkwormSite(new Coor3D(0, 50, 0), 25, Side.RED);
    SilkwormSite site3 = new SilkwormSite(new Coor3D(10, 40, 0), 25, Side.RED);
    r.addSite(site1);
    r.addSite(site2);
    r.addSite(site3);
    Ellipse aou = new Ellipse(new Coor2D(0, 0), 0.0, 35.0, 25.0);
    FireMission f1 = new FireMission(60.0, numberOfMissiles, new Coor2D(0, 0), aou);
    site1.setMission(f1);
  Schedule.clearRerun();
  Schedule.setSingleStep(false);
  Schedule.setVerbose(false);
  Schedule.stopOnTime(600.0);
  Schedule.startSimulation();
  int hits;
  int homes;
  int shots;
  for (int shipNumber = 0; shipNumber < ship.length; shipNumber++) {
    BasicModSimComponent thisShip = ship[shipNumber];
    hits =  ((Integer) thisShip.getProperty("Hits", new Integer(0))).intValue();
    homes = ((Integer) thisShip.getProperty("HomedBy", new Integer(0))).intValue();
    shots = ((Integer) thisShip.getProperty("MissileShots", new Integer(0))).intValue();
    hitCounter[shipNumber].newObservation(hits);
    homesCounter[shipNumber].newObservation(homes);
    shotsCounter[shipNumber].newObservation(shots);
  }
  missilesShotDown.newObservation(tab.getMissilesKilledByMissiles());
  missilesPassive.newObservation(tab.getMissilesKilledByPassive());
  missilesFuel.newObservation(tab.getMissilesKilledByFuel());
  screen.reset();
  r.reset();
  site1 = null;
  site2 = null;
  site3 = null;
  r = null;
  screen = null;
  Schedule.reset();
}
System.out.println("Angle of attack " + angle);
System.out.println("Averages by ship");
System.out.println(" " + '\t' + '\t' + "HITS by Enemy Missiles" + '\t'
                + '\t' + "HOMING by Enemy Missiles" + '\t'
                + '\t' + "SHOTS at Enemy Missiles" + '\t'
                + '\t' + '\t' + "Enemy missiles Killed"  );
System.out.println("Name" + '\t' + '\t'
  + "Mean" + '\t' + "Std Dev" + '\t' + "Max" + '\t' + "Min" + '\t'
  + "Mean" + '\t' + "Std Dev" + '\t' + "Max" + '\t' + "Min" + '\t'
  + "Mean" + '\t' + "Std Dev" + '\t' + "Max" + '\t' + "Min" + '\t'
  + '\t' + "Mean" + '\t' + "Std Dev" + '\t' + "Max" + '\t' + "Min");
for (int shipNumber = 0; shipNumber < ship.length; shipNumber++) {
  String name = ship[shipNumber].getName();
  String msg = name + '\t' + '\t';
  msg = msg + nf.format(hitCounter[shipNumber].getMean()) + '\t';
  msg = msg + nf.format(hitCounter[shipNumber].getStandardDeviation()) + '\t';
  msg = msg + nf.format(hitCounter[shipNumber].getMaxObs()) + '\t';
  msg = msg + nf.format(hitCounter[shipNumber].getMinObs()) + '\t';
  msg = msg + nf.format(homesCounter[shipNumber].getMean()) + '\t';
  msg = msg + nf.format(homesCounter[shipNumber].getStandardDeviation()) + '\t';
  msg = msg + nf.format(homesCounter[shipNumber].getMaxObs()) + '\t';
  msg = msg + nf.format(homesCounter[shipNumber].getMinObs()) + '\t';
  msg = msg + nf.format(shotsCounter[shipNumber].getMean()) + '\t';
```

```
          msg = msg + nf.format(shotsCounter[shipNumber].getStandardDeviation()) + ‘\t’;
          msg = msg + nf.format(shotsCounter[shipNumber].getMaxObs()) + ‘\t’;
          msg = msg + nf.format(shotsCounter[shipNumber].getMinObs()) + ‘\t’;
          if (shipNumber == 1) {
            msg = msg + "Missile" + ‘\t’ ;
            msg = msg+ nf.format(missilesShotDown.getMean()) + ‘\t’;
            msg = msg + nf.format(missilesShotDown.getStandardDeviation()) + ‘\t’;
            msg = msg + nf.format(missilesShotDown.getMaxObs()) + ‘\t’;
            msg = msg + nf.format(missilesShotDown.getMinObs()) + ‘\t’;
          }
          if (shipNumber == 2) {
            msg = msg + "Passive" + ‘\t’;
            msg = msg+ nf.format(missilesPassive.getMean()) + ‘\t’;
            msg = msg + nf.format(missilesPassive.getStandardDeviation()) + ‘\t’;
            msg = msg + nf.format(missilesPassive.getMaxObs()) + ‘\t’;
            msg = msg + nf.format(missilesPassive.getMinObs()) + ‘\t’;
          }
          if (shipNumber == 3) {
            msg = msg + "Fuel" + ‘\t’;
            msg = msg+ nf.format(missilesFuel.getMean()) + ‘\t’;
            msg = msg + nf.format(missilesFuel.getStandardDeviation()) + ‘\t’;
            msg = msg + nf.format(missilesFuel.getMaxObs()) + ‘\t’;
            msg = msg + nf.format(missilesFuel.getMinObs()) + ‘\t’;
          }

          System.out.println(msg);
        }
        for (int aa = 0; aa < 4; aa++) {
          hitCounter[aa].reset();
          homesCounter[aa].reset();
          shotsCounter[aa].reset();
        }
        angle = angle + 5;
      }
      numberOfMissiles = numberOfMissiles + 10;
    }
    System.exit(0);
  }
}
```

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center……………………………………………….2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library……………….. ……………………………………………….2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Professor James G. Taylor……………………………………………………………5
   Code OR
   Naval Postgraduate School
   Monterey, California 93943-5002

4. Professor Arnold H. Buss……………..……………………………………………….2
   Code OR/SB
   Naval Postgraduate School
   Monterey, California 93943-5002

5. Professor Gordon H. Bradley..………..……………………………………………….1
   Code OR/BZ
   Naval Postgraduate School
   Monterey, California 93943-5002

6. Wayne P. Hughes…………………...……………………………………………….2
   Code OR/
   Naval Postgraduate School
   Monterey, California 93943-5002

7. LCDR James R. Townsend………...………………………………………………….3
   Naval Undersea Warfare Center
   Newport, Rhode Island 02841-1708